UNIVERSITETET I OSLO
Institutt for informatikk

Kan sosiale nettsteder benyttes til opplæring av nevrale nettverk?

Masteroppgave
60 studiepoeng

Steinar Hamdahl

1. november 2007

Figurer

Tabeller

Takk til

Jeg vil benytte anledningen til å si tusen takk til Steinar Kristoffersen for all tiden han har brukt på meg og denne oppgaven. Jeg har virkelig satt pris på alle samtalene våre og har alltid fått ny giv til å skrive etter å ha snakket med deg! Vil også takke for at jeg har fått lov til å fokusere på et emne jeg finner interessant.

A special thanks to Maia Dimitrova! You are the only one I've gotten in contact with that has been able to answer my technical questions about neural networks. And you have done so far beyond my hopes and expectations! This paper would not have been the same without you.

Hjertelig takk til Harald Furu for god støtte, nyttige råd og fremdragende coaching!

Tusen takk til Olav Amlie - jeg hadde aldri ferdigstilt noen mastergrad uten deg og din hjelp!

Takk til bibliotekarene på institutt biblioteket - dere er både hjelpsomme, hyggelige og faglige dyktige langt utover det som kan forventes av dere og fortjener en kjempe takk!

Takk også til Henrik Haaberget, Jon Sagberg, Bjørn Erik Deverill-Mathisen, Ellen Munthe-Kaas, Rolv Kronen, Anna Lekova, Jo Herstad, Herman Ruge Jervell, Monica Holter og Jim Tørresen som hver på sin måte har bidratt til denne oppgaven.

Ellen og Victoria skal også ha tusen takk for på hver sin måte og etter beste evne å utvise tålmodighet og støtte. En spesiell takk til min mor for barnepass hvis foruten hadde gjort dette enormt mye vanskeligere å fullføre!

Oslo, 30. oktober 2007

Steinar Hamdahl

Sammendrag

Jeg presenterer i denne oppgaven en ny måte å danne grunnlag for opplæring av nevrale nettverk på, nemlig ved hjelp av sosiale nettsteder med tilhørende tagge kultur (folksonomier) og eventuell støtte i taksonomier. Jeg viser gjennom forskning og enkle, egenproduserte applikasjonseksempler at det vil være mulig ved nevnte fremgangsmåte å produsere fullt anvendelige nevrale nettverk.

Jeg søker videre å vise ved relevant forskning at fungerende nevrale nettverk burde benyttes utover originalt tiltenkt funksjonalitet.

1 Innledning

1.1 Bakgrunn

I denne oppgaven vil jeg vurdere mulighetene til å bruke internettsamfunn til trening av databaserte gjenkjennelsesalgoritmer.

Bruk av datamaskiner til gjenkjenning av objekter brukes i dag for å kunne drive vidt forskjellige automatiserte prosesser som flaskepanting, papirgjenvinning, fingeravtrykksgjenkjenning, musikkgjenkjenning, ansiktsdeteksjon for autofokus i digitalkameraer, og så videre. Allikevel utføres det store mengder med manuelt gjenkjenningsarbeid. Ta som eksempel billedkategorisering; hvor mange har vel ikke brukt timevis på å kategorisere den digitale billedsamlingen sin men allikevel slitt for å finne ett bilde av akkurat *den tingen* i etterkant? Mye av dette manuelle arbeidet finnes det per i dag løsninger til å kunne utføre maskinelt, for eksempel å sortere ut portrett bilder.

Målet med denne oppgaven er å se nærmere på om det finnes et, til nå, ikke utnyttet potensial til å la maskinlæringsalgoritmer (”machine learning algorithms”), som for eksempel nevrale nettverk, ta seg av en større del av kategoriserings- og gjenfinningsarbeid av multimedia informasjon. For at slike algoritmer skal kunne gjøre slikt arbeid må de først læres opp til det – derav maskin*lærings*algoritmer. Spørsmålet er først og fremst hvordan man kan få utført denne læringen. Og nærmere bestemt vil jeg se om én kan forbedre læringen og (dermed) ytelsen ved å integrere slike kategoriseringsalgoritmer i sosiale strukturer som internettsamfunn. Læring og trening er ord jeg bruker om hverandre og betegner akkurat det samme, nemlig gradvis tilpasning av en algoritme til å gi et ønsket svar utifra et gitt problem.

Grunnlaget for ønsket om å se nærmere på integrering av maskinlæringsalgoritmer i nettopp sosiale strukturer er fordi slike algoritmer som regel trenger svært omfattende opplæring for å kunne fungere optimalt. Faktisk trenger algoritmer basert på nevrale nettverk omfattende opplæring for å kunne fungere i det hele tatt. Slik opplæring må utføres ved å vite hva som mates inn i en slik algoritme slik at en i etterkant kan korrigere algoritmens svar som enten feil eller riktig. En snakker gjerne om mating av fra tusenvis til titusenvis av kjente eksempler før en slik algoritme begynner å fungere tilfredsstillende. Og dette kreves for hver type objekt én ønsker at algoritmen skal gjenkjenne. Altså trenger man en kilde som kan levere multimedia

materiale av kjent karakter i relativt store mengder. For ytterligere å forbedre ytelsen, må algoritmens brukere også i etterkant av å ha oppnådd en tilfredsstillende ytelse kunne fortsette å korrigere algoritmen. Et naturlig testområde for slik bruk faller da på sosiale nettsteder som Flickr.com eller YouTube.com hvor det allerede finnes store mengder multimedia data og det hele tiden er et tilsig av nye data. I tillegg finnes det, i hvert fall teoretisk sett, mange brukere som er i stand til manuelt å kunne korrigere en slik algoritme.

Tematiske sosiale nettsteder har gjerne en form for kategorisering tilgjengelig for brukeres data allerede i utgangspunktet. Dette kan være i form av ”tags” (annoteringer), taksonomier eller annet. Slik kategoriseringsinformasjon kan muligens benyttes for automatisert opplæring av nevrale nettverk og andre maskinlærings algoritmer. For eksempel benytter Flickr.com seg av ”tags” til kategorisering av opplastede bilder. Forskning hevder at ved et stort nok antall ”tags” på et element oppstår såkalte ”wise-tags” (fornuftige merkelapper). Tar vi med at for eksempel Flickr.com har et fritt tilgjengelig programmerbart API, betyr det muligens at Flickr.com allerede i sin nåværende form er mulig å benytte til tilfredsstillende opplæring av maskinlæringsalgoritmer.

Når én har lært opp en gjenkjennelsesalgoritme på nevnte måte, vil jeg videre argumentere for at én burde gå videre og også benytte algoritmen i andre implementasjoner enn opplæringsmiljøet. Et naturlig steg er da å gå over i domenet dataminering. Der finnes det allerede tilgjenglig et Extended Markup Language (XML) basert språk kalt Predictive Model Markup Language (PMML) for overføring av datamineringsmodeller, deriblant nevrale nettverk og andre utvalgte maskinlæringsalgoritmer. PMML er laget av en organisasjon kalt The Data Mining Group (DMG) som er et konsortium av mange store programvare leverandører som blant andre IBM, Microsoft, Oracle, SAS og SPSS. Flere av disse leverandørene har implementert støtte for PMML i sine databasesystemer. Altså burde opplæring og bruk av en maskinlæringsalgoritme i et (oppstarts-) miljø kunne være en av fasene til brukssyklusen til algoritmen.

1.2 Problemstilling

Hvordan kan en mest mulig effektivt trene opp og benytte seg av nevrale nettverk til å utføre kategoriseringsarbeid på multimedia materiale?

Det finnes svært mange forskjellige typer maskinlæringsalgoritmer og det ville blitt alt for omfattende å ta for meg alle forskjellige former her. Jeg velger derfor å konsentrere meg om nevrale nettverk (ANN – Artificial Neural Network). Dette delvis fordi en del forskning ([29], [30], [41]) fremstiller

nevrale nettverk som den type gjenkjennelses- og beslutningsalgoritme som har et potensial til å oppnå høyest ytelse under komplekse forhold. Med komplekse forhold menes at informasjonen som søkes igjennom ikke er optimalisert og kan inneholde mindre eller mer informasjon enn det søkes etter samt mulig støy. Viktig er det også at ytelsesevnen tilsynelatende kan fortsette å øke lenger ved ytterligere opplæring enn ved andre typer maskinlæringsalgoritmer. Mer ”statisk” utformede maskinlæringsalgoritmer, som for eksempel Bayesisk nettverk (Bayesian network), tenderer å flate ut med hensyn til ytelse etter en gitt mengde opplæring. Til gjengjeld krever slike algoritmer mindre opplæring/oppsamlede data enn algoritmer basert på nevrale nettverk. ([41], [1], [7])

Uansett utforming må maskinlæring algoritmer ”læres” opp. Og grunnlaget for opplæringen er hovedpunktet i denne oppgaven. Det er ingen ting som tilsier at prinsippene omtalt i denne oppgaven med nevrale nettverk som utgangspunkt, ikke også vil være gyldige for andre typer maskinlæringsalgoritmer.

Jeg vil forøvrig gjøre oppmerksom på at kunstige nevrale nettverk siden det først ble foreslått på 1960-tallet har hentet inspirasjon fra biologiske nevrale nettverksmodeller. Det betyr derimot ikke at de er like verken i struktur eller egenskaper som biologiske nevrale nettverk. Biologiske nevrale nettverk er atskillig mer komplekse og fungerer på en helt annen måte og med helt andre forutsetninger og målsetninger enn kunstige nevrale nettverk ([37]). Jeg vil nærmere beskrive hvordan nevrale nettverk som data algoritmer bygges opp i kapitlet ”nevrale nettverk”.

I denne sammenheng har jeg lyst til å nevne at medieomtalen (bbc.co.uk og gjengitt av engadget.com, ITavisen.no, og mange, mange flere) rundt [43], fremstilte prosjektet som om det forsøkte å simulere en (halv) musehjerne. Dette var slettes ikke tilfelle. Det som ble simulert var et nevralt nettverk som med hensyn til størrelse og kompleksitet tilsvarer det man er ganske sikker på er omlag 80% av en halv musehjernes kapasitet. Men funksjonaliteten og effektiviteten – eller ”intelligensen” om man vil – til et nevralt nettverk, ligger ikke alene i størrelse og kompleksitet. Det ligger i en blanding av mange faktorer som struktur, læring, kompleksitet og størrelse. I [43] var verken struktur eller noen form for læring fra en musehjerne forsøkt implementert. Dermed kan det ikke hevdes at [43] på noen som helst måte simulerte en musehjerne. Jeg ser på dette eksempelet som en relativt vanlig og uheldig sammenblanding og misforståelse av biologiske og kunstige nevrale nettverk. Jeg vil i kapittel to gi en beskrivelse av hva kunstige nevrale nettverk er og hvordan de fungerer, samt svært grunnleggende gi en sammenlikning med hvordan kognitiv psykologi definerer biologiske nevrale nettverk. Senere i denne oppgaven vil jeg ofte skrive kun nevrale nettverk og vil da mene kunstige nevrale nettverk.

Tilbake til problemstillingen. Denne kan sees som bestående av to elementer; utvikling og bruk. For ”normale” algoritmer vil disse gjerne kunne holdes fra hverandre; man utvikler en algoritme til et produkt som så kan brukes. For maskinlæringsalgoritmer er det nødvendigvis ikke slik. For slike algoritmer vil det gjerne være en gjensidig avhengighet mellom utvikling og bruk, ikke minst algoritmer baserte på nevrale nettverk. Dette fordi de ikke utvikles til et statisk funksjonsnivå, men hele tiden dynamisk utvikles og tilpasses via tilbakemeldinger til algoritmen. Altså; så lenge mottatt ny informasjon ikke er repeterende og/eller funksjonsnivået ikke er optimalisert må brukerne kunne gi tilbakemeldinger til systemet som benyttes, hvilket igjen fordrer aktivt deltagende brukere. Dermed settes det visse begrensninger for i hvilke tilfeller det vil være hensiktsmessig å benytte slike algoritmer. Samtidig er dette et aspekt som gjør slike algoritmer attraktive i visse tilfeller, nemlig der hvor potensiell fremtidig informasjon ikke kan defineres på forhånd.

Som jeg vil vise gjennom presentert forskning, for eksempel [32], er nevrale nettverk avhengig av ekstensiv opplæring. Og det er ikke nødvendigvis ferdig utlært etter en bestemt mengde opplæring da nevrale nettverk som regel blir bedre og bedre desto mer opplæring de blir gitt. Hvorfor skulle en så ønske å benytte seg av dem hvis de krever så mye og kanskje aldri gir perfekte svar uansett? Problemet med opplæring er, som allerede nevnt, også det som gjør nevrale nettverk ønskelig å bruke; de ”skreddersys” til problemstillingen nettopp av opplæringen. Der andre algoritmer må spesialutvikles med ekspertkunnskap for hver enkelt oppgave kan nevrale nettverk tas i bruk såfremt man har nok data å mate med. Ta for eksempel en gjenkjenningsalgoritme for stemmer; det går fint an å lage en statisk algoritme som kjenner igjen én stemme. Men hva skjer dersom en annen bruker med en annen stemme med et annet stemmeleie ønsker å ta i bruk en slik algoritme? Og hva hvis en med en annen dialekt ønsker å ta den i bruk? Og hva når språk utvikler seg med nye meninger for ord, eller helt nye ord for den saks skyld? En slik algoritme vil for alltid kunne bli utsatt for ny informasjon. Altså trengs en dynamisk algoritme som kan utvikle seg sammen med strømmen av data. Da er nevrale nettverk eller andre typer maskinlæringsalgoritmer en løsning.

Hvorfor er så ikke slike løsninger mer utbredt? Her er det nok flere faktorer som spiller inn. Jeg vil i denne oppgaven gå igjennom relevante faktorer i henhold til problemstillingen. Følgende er et sammendrag av det jeg ser på som sentrale faktorer i denne sammenheng og som jeg senere vil gå igjennom mer detaljert.

Kravet til opplæring er selvfølgelig en viktig faktor. Et nevralt nettverk kan ikke tas i bruk med en gang. Mengden trening som trengs vil variere med oppgaven det nevrale nettverket skal løse, men det er gjerne behov for mange tusen både positive og negative eksempler før et nevralt nettverk kan begynne å fungere tilfredsstillende. Med positive eksempler menes det en vil det nevrale nettverket skal kjenne igjen, mens negative skal gi negativt resultat ved forsøk på gjenkjennelse. ([29], [33], [41], [1], [7])

Et annen vesentlig problem er de kapasitetsmessige kravene som fort kan bli høye både med hensyn til regnekapasitet og hukommelse. Dette kan også variere kraftig avhengig av type nevralt nettverk og tilhørende metoder som velges. Et eksperimentelt faktum er at jo mer effektiv metode desto større krav til tilgjengelig minne. ([29], [33], [41], [1], [7])

Nevrale nettverk fungerer ved hjelp av parallell prosessering mens datamaskiners prosesseringsenheter normalt fungerer etter Von Neumann arkitektur, det vil si sekvensiell. Normale datamaskiner er altså ineffektive i behandling av nevrale nettverk.

Det er store forskjeller på nevrale nettverk metodologier med hensyn til bruksområde. Et spesifikt problem innen bruk med bilder er at det raskt er et veldig stort antall innparametre. Der et system for aksjeanalyse kan klare seg med under ti forskjellige innparametre vil et bilde ha fra mange hundre til mange millioner innparametre. Konsekvensen er enten svært store krav til minne- og prosesseringskraft eller spesialiserte løsninger.

Et trent nevralt nettverk utgjør en unik og lukket enhet. Informasjon og funksjonalitet tilegnet et nevralt nettverk ligger gjemt i selve strukturen dens og kan ikke på noen enkel måte utnyttes eller videreutvikles på noen annen måte enn bruk av det bestemte nevrale nettverket. Skal en dele på funksjonaliteten til et nevralt nettverk finnes det per i dag altså to valg:

- Å dele kopier av det nevrale nettverket. Kan gjøres ved direkte kopi eller eksport av modellen ved hjelp av datamineringsspråk som PMML.
- Å gjøre en instans av det nevrale nettverket tilgjengelig for alle aktuelle parter. Dette kan selvfølgelig gjøres ved hjelp av en server tjeneste, eller såkalt grid teknologi.

Uansett valgt løsning må alle brukere stå i direkte kontakt med hverandre.

Ny forskning derimot ser ut til å kunne klare å kombinere flere trente nevrale nettverk til ett. Hvis man klarer å få til dette vil man kunne utføre distribuert opplæring. Men først når slik distribuert opplæring kan gjøres på heterogent datagrunnlag kan man begynne å snakke om ekte distribuert opplæring. Må

datagrunnlaget være homogent vil det bare være snakk om en form for grid teknologi. Dessverre ser det ut til at dagens forskning ikke har kommet lenger enn til distribuert opplæring med homogent datagrunnlag. [6]

Disse problemene betyr ikke at kunstige nevral nettverk ikke brukes. Nevrale nettverk er mye brukt i autonom styring, finans analyse- og prediksjonsprogramvare, data minering, klassifisering, prediksjon, beslutningstagning i for eksempel spill, meteorologi, og så videre. LeCun et al. hevder sågar at nevrale nettverk er så godt som enerådende som klassifiserings algoritmer innen tekstgjenkjenning (OCR). ([41], [3])

For å eksemplifisere problemstillingen har jeg valgt å avgrense til kun billedbruk for egne eksempler. Dette har også farget av på hvilke modeller og metodikker jeg har sett nærmere på i denne oppgaven.

1.3 Annen forskning

Mange aspekter og problemer ved å bruke nevrale nettverk er godt belyst gjennom forskning. Jeg vil nå gå igjennom relevante fokusområder i henhold til denne oppgaven.

1.3.1 Bruk av nevrale nettverk på billedmateriale

LeCun et al. (1998) omhandler tegngjenkjenning og tar som inndata et bilde på 38x38 billedpunkter. De fremhever at til bruk i automatisk gjenkjenning i bilder vil bildene gjerne være store – opptil flere hundre variabler. Med variabler menes her billedpunkter. Det resulterende nevrale nettverket etter LeCun et. al sin modell – et såkalt ”convolutional” (oversatt; innviklet/sammenrullet) nevralt nettverk som jeg skal forklare nærmere senere – har 340´908 koblinger. Riktignok er noe av poenget med det spesifikke “convolutional” nevralt nettverk modellen at den reduserer antall koblinger som må kalkuleres. I nevnte tilfelle blir de 340´908 koblingene redusert til 60´000 veide, trenbare koblinger. Tar vi dette til 2007 er det heller mange millioner variabler enn mange hundre i et ordinært bilde. (”1 mega piksel” ≅ en million individuelle variabler og 3Mp er per i dag relativt vanlig – i mobiltelefoner). I henhold til LeCun et al. vil et fullt tilkoblet første lags nevralt nettverk med hundre noder for inndata tradisjonelt tilsi flere titalls tusen med koblinger. Økningen av innvariable fører til noe i overkant av en kvadratisk økning av det nevrale nettverket for avanserte nevrale nettverksmodeller som ved ”convolutional neural network” eller pyramidiske nevrale nettverk ([41], [1]). Ved enklere nevrale nettverksmodeller som rene FNN (Feed forward Neural Network), blir denne økningen nærmere eksponentiell. Dette tilsier fort meget store krav til maskinvare.

Løsningen på dette problemet har tradisjonelt vært å lage en algoritme som foretar en mønster gjenkjenning (”feature extraction”) av predefinerte mønstre. Disse mønstrene representeres gjerne i en eller annen form for geometriske vektorer. Dette reduserer et stort antall variable til et lite og ikke minst kjent antall tall som kan benyttes som inndata. Selve det nevrale nettverket kan dermed gjøres lite og raskt. Problemet med denne fremgangsmåten er, som LeCun et al. fremhever, at mønstergjenkjenningsmodellen lages på bakgrunn av ekspertkunnskap, og må spesialdesignes for en type objekt av gangen. Det resulterende systemet blir derved låst til den eller de oppgavene eksperten(e) initielt så for seg, og ytelsen blir avhengig av kompetansen til eksperten(e). Altså vil en i mange slike tilfeller ikke tjene noe på å gå for et nevralt nettverk fremfor statistisk baserte metoder som for eksempel clustering.

1.3.2 Læring (også kalt trening)

Et av de grunnleggende problemene jeg ønsker å se nærmere på, er å få eksponert et nevralt nettverk for så mye potensiell opplæring som mulig. For å effektivisere den innledende opplæringen som bringer algoritmen opp på et nivå hvor den kan tas i bruk, er flere former for såkalt ”bootstrapping” foreslått. [29] og [30] sin løsning på slik ”bootstrapping” er å ta to sett med bilder hvor det ene settet består av bilder med innholde de ønsker gjenkjennet, mens sett nummer to består av bilder de ikke ønsker gjenkjennet. Deretter femtendobler de antall bilder ved å gjøre små endringer på dem som å rotere dem litt eller endre utstrekning eller annen enkel transformasjon. Det ”negative” settet med bilder – altså bildene med uønsket innhold – laget de simpelthen ved å generere støy (billedpunktene ble satt til vilkårlige verdier). Resultatet ble altså at de ut ifra et relativt lite sett med bilder fikk et stort sett med kjente data og matet det nevrale nettet med. Deretter repeterte de mating med disse dataene inntil det nevrale nettet gav tilfredsstillende lite feil tilbakemeldinger.

Et mål med den initielle opplæringen vil altså være å sørge for at den skjer kontrollert men automatisk siden algoritmen ikke vil være brukelig før denne prosessen er over.

En annen foreslått måte å oppnå ekstensiv opplæring på er å distribuere algoritmen ut på “Griden” ([36], [38]). Det betyr at en vil kunne dekke eventuelle behov til prosesserings- og lagringskapasitet. Det kan også være muligheter for å dekke delt benyttelse av den aktuelle algoritmen på denne måten. I så tilfelle vil flere kunne delta i opplæring av algoritmen noe som kan være en løsning på det grunnleggende problemet om nok opplæring.

Som allerede nevnt ser det ut til at helt ny forskning kan kombinere flere individuelt opplærte nevrale nettverk til ett. Dette betyr muligheten for distribuert lokal trening av et nevralt nettverk. Dessverre har positive resultater

kun vært mulig å oppnå ved å benytte det samme grunnlaget for trening. Altså vil en slik løsning foreløpig ikke være til noen særlig nytte. ([6])

1.3.3 Deling og kollaborering

Et problemet med [38] og til dels [6] sine løsninger er at de ikke adresserer sikring av brukermasse og dermed selve opplæringsgrunnlaget. Det er jo ikke nødvendigvis slik at et system benyttes bare fordi det er tilgjengelig. Griden sørger kun for tilgang på ressurser, ikke nødvendigvis brukere.

Samme problem som [38] vil forøvrig gjøre seg gjeldende hvis en oppretter et sosialt nettsted med den hensikt å få opplæringsgrunnlag til et nevralt nettverk; hvordan skal en få den nødvendige brukermassen til å gjøre det hensiktsmessig?

Dette kan løses hvis man har kontroll over en sosial struktur. For eksempel at man kontrollerer en organisasjon sitt intranett og kan pålegge organisasjonens medlemmer å benytte funksjonalitet på en grid struktur.

Den opplagte løsningen for å sikre brukere, mener jeg, må være å implementere en slik algoritme i et allerede eksisterende sosialt nettverk, som for eksempel Flickr.com, hvor medlemmene allerede benytter den funksjonaliteten man ønsker. Det er ikke dermed sagt at et slikt system kan implementeres i det eksisterende sosiale nettverket. Altså blir [38] sin løsning viktig, ikke for å sikre brukere men for å la brukerne benytte en og samme instans av det nevrale nettet.

1.4 Mangler ved annen forskning

1.4.1 Internettsamfunn og nevrale nettverk

Inspec ([139]) er en referansebase over vitenskaplige artikler fra blant andre IEEE og ACM, og søk her kommer frem med tusenvis av artikler om nevrale nettverk, maskinlæringsalgoritmer, anvendelser av slike algoritmer, samt hundrevis av artikler om tagging, taksonomier, folksonomier og internettsamfunn. Men jeg har ikke funnet noen vitenskapelige forsøk som søker å benytte et sosialt nettsted som basis for opplæring av et nevralt nettverk.

Inspec søk returner riktignok treff som beskriver bruk av sosiale nettsteder sammen med nevrale nettverks løsninger som en del av nettstedets funksjonalitet. To treff omhandler bruk av nevrale nettverk i sikkerhetsfunksjonaliteten til spesifikke nettsteder. De resterende omhandler bruk av nevrale nett til å predikere nettbrukeres vaner. Altså benyttes nevrale nettverk i disse tilfellene som verktøy for eller i internettsamfunn og ikke internettsamfunn som verktøy for nevrale nettverk i disse tilfellene.

Jeg har også vært i samtaler med Associate Professor Maya Dimitrova og Associate Professor Anna Lekova ved Institute of Control and System Research, Bulgarian Academy of Sciences, samt førsteamanuensis Ellen Munthe-Kaas, universitetslektor Gisle Hannemyr, Professor Herman Ruve Jervell og Professor Jim Tørresen ved institutt for informatikk, Universitet i Oslo. Ingen av disse personene har hørt om noe prosjekt som har benyttet sosiale nettsteder til opplæring av nevrale nettverk.

Dette betyr selvfølgelig ikke at min idé ikke har vært tenkt og kanskje til og med blitt utført tidligere. For eksempel har Google kjøpt opp et selskap som heter Neven Vision og som har spesialisert seg på billedanalyser blant annet ved hjelp av nevrale nettverk. Dette oppkjøpet har gitt resultater slik at man via en applikasjon som per i dag finnes i beta versjon kan søke etter bilder på internett ved hjelp av mønster beskrivelser som for eksempel ”ansikt” eller ”sykkel”. Ikke utenkelig at Neven Vision benytter seg av informasjon hentet fra, om ikke sosiale nettsteder så i hvert fall, internettsteder til opplæring av sitt system. Patent søknader levert inn av Neven Vision tilsier derimot at deres nevrale nettverksløsning benytter mønstergjenkjenning i forprosesseringen. Dette vil jeg komme nærmere tilbake til betydningen av i kapittelet om hva nevrale nettverk er, men konsekvensen er at det nevrale nettverket ikke er opplært til å kjenne igjen objekter. I slike systemer skal det nevrale nettverket avgjøre om en på forhånd utviklet representasjon av et objekt eksisterer i en representasjon av et bilde. Dermed vil uansett ikke Neven Vision sin teknologi være optimal for automatisk opplæring ved hjelp av nettsteder siden selve objekt definisjonen i større eller mindre grad må lages manuelt. Det vil si at det på forhånd bestemmes hva det nevrale nettverket skal kunne kjenne igjen. Men dette blir altså til en viss grad spekulasjoner siden Neven Vision sin teknologi i helhet samt full beskrivelse av faktisk bruk er hemmeligholdt.

1.4.2 Mønster gjenkjenning

Et svært viktig moment for gjenkjenningsalgoritmer er selvfølgelig gjenkjennelse av enkelt mønstre (feature extraction). LeCun et al. [41] skriver som tidligere nevnt at den tradisjonelle mønster gjenkjenningen normalt har vært en statisk algoritme basert på ekspertkunnskap. Videre mener de at disse ekspertene antageligvis neppe klarer å få med all relevant informasjon fra inndata. LeCun et al. foreslår i stedet å la nevrale nettverk ta seg av hele gjenkjennelsesprosessen mer likt slik biologiske nevrale nettverk gjør det.

[37] på sin side oppfordrer til varsomhet når en henter inspirasjon fra nevrologisk forskning og kognitiv psykologi. De mener at målene og midlene til disse vitenskapene og multimedia gjenfinning innen IT er svært forskjellige og at den overlappingen som finnes mellom fagfeltene kun egner seg til inspirasjon.

1.4.3 Ytelsesmåling

Et generelt problem ved gjenkjennelses algoritmer er å måle ytelsen i forhold til hverandre. Det finnes flere testsett med bilder tilgjengelig, men ingen felles konsensus om hvilket som skal benyttes. Dessuten har gjerne hvert enkelt forsøk sin egen vinkling av en problemstilling og kan derfor ofte ikke sammenliknes direkte med andre løsninger.

1.4.4 Tagging

[33] hevder at taggesystemer er lite studert eller forstått. Dette kan jeg støtte opp om ettersom jeg har funnet lite relevant forskning om temaet. Likeledes står det til med taggeskyer i henhold til [82]. Tagging og medfølgende teknologi er altså i en startfase vitenskapelig sett. Siden det likevel er tatt i bruk i stor skala og det etterlyses mer forskning på fenomenet regner jeg det som gitt at tagging som sådan vil fortsette å bli benyttet i fremtiden. Derfor anser jeg det som trygt å basere et system delvis på andre (nettsteders) fortsatte bruk av slik teknologi.

1.5 Bidrag

Jeg ønsker å se om det finnes nye måter å lære opp nevrale nettverk på, som forhåpentligvis vil være så effektiv at de resulterende gjenkjennelses-algoritmene kan benyttes utover det opprinnelige formålet.

Løsningen mener jeg er å lage applikasjoner som kan benyttes samtidig som det trener opp slike algoritmer, og å bringe slike applikasjoner ”ut til folket”. For å gjøre det må man finne passende applikasjonsområder. Jeg tror at bilde-gjenkjenning/kategorisering vil være et relativt enkelt sted å begynne fordi det allerede finnes massive mengder fritt tilgjengelig via internett.

Når en så har en ferdig trent algoritme bør denne benyttes videre andre steder og kanskje til og med til andre formål enn kun ved det originale opptrenings sted. Til dette kan en for eksempel benytte PMML, et XML basert språk laget for å kunne eksportere datamineringsmodeller. Jeg ser altså for meg at man lærer opp en algoritme til et visst minstemål for gjenkjenning av objekter via et sosialt nettsted. Deretter kan det deles med frittstående brukere. Det ultimale må være at alle brukere videre vil kunne bidra til videre opplæring ved kollaborering av videreutviklede utgaver av algoritmen. Denne videre opplæringen bør kunne deles ved hjelp av modellene og ikke nye data.

Som eksempel: Et sosialt nettsted interesserer seg for dyrearter. En forsker som skal inn i en regnskog hvor kontakt med internett vil være umulig, laster ned gjenkjennelsesalgoritmen for dyrearter opplært av det sosiale nettstedets data og/eller brukere. Algoritmen lastes inn på et prosesseringskapabelt kamera (for

eksempel en mobiltelefon) slik at kameraet kan gi svar med en gang om et bilde inneholder en - for algoritmen – kjent dyreart. Forskeren korrigerer hvis feil art blir rapportert og legger eventuelt til ny informasjon. Ved tilgang til internett lastes den lille, i forhold til bildene, nevrale nettverksmodellen opp til det sosiale nettstedet hvor de kombineres til et nytt og oppdatert nevralt nettverk.

1.6 Metode

Det finnes en rekke praktiske problemer ved å la et sosialt nettsted, eller liknende, lære opp nevrale nettverk. Her følger de jeg har funnet frem til:

- Vil data fra et sosialt nettsted ha nok mangfold med hensyn til antall kategoriseringer til at det nevrale nettverket får tilfredsstillende funksjonalitet?
- Kreves det bestemte typer sosiale nettsteder, eventuelt - må nettstedene benytte spesifikke kategoriseringsmetoder, for å få et slikt mangfold?
- Kan man få nok data fra et sosialt nettsted for hver enkelt kategori som skal læres til å kunne trene opp et nevralt nettverk til et tilfredsstillende funksjonsnivå? Dette innebærer at det for hver kategorisering må kunne finnes noe sånt som tusen ensartede enheter. Dette kravet kan senkes betydelig ved bruk av avanserte nettverksstrukturer og læringsmetoder, men det må fortsatt være et betydelig antall enheter bak hver enkelt kategorisering. ([30])
- Har man per i dag nok maskinkraft til å kunne drive et slikt system?

Jeg vil gå igjennom forskningslitteratur for finne ut av disse spørsmålene. Jeg vil forsøke å finne svar på teoretiske spørsmål som:

- Hva slags kategoriseringsmetode er meste egnet brukt ved et sosialt nettsted for å løse problemstillingen?
- Hva slags nevralt nettverk er best egnet til å løse problemstillingen på beste mulig måte og samtidig overvinne de praktiske problemene?
- Finnes det datakraft nok til å drive slike systemer?
- Finnes det teoretiske løsninger for å løse ellers uløselige krav til maskinvare?

Jeg vil altså som metode for å finne løsning på problemstillingen, gå igjennom forskningslitteratur for å se om denne kan gi tilfredsstillende svar.

For å teste ut mine resultater har jeg laget et system som benytter Flickr til å lære opp et nevralt nettverk til å gjenkjenne innhold i bilder. Årsaken til at jeg ønsker å teste det ut er at jeg ikke har funnet frem til noen som har forsøkt å

lære opp nevrale nettverk på akkurat denne måten. Da gjenstår å bekrefte eller avkrefte om det er mulig ved å teste ut selv.

I et slikt eksperiment anser jeg to aspekter å være spesielt interessante:

- Hvor komplekst kan det nevrale nettverket gjøres med hensyn på inndata (antall billedpunkter) og utdata (antall mulige kategoriseringer) og fremdeles benyttes tilfredsstillende på en ”ordinær” datamaskin? Med ”ordinær” mener jeg en datamaskin som kan antas å være representativ for den jevne Flickr bruker.
- Hvor mange bilder må til i opplæringen av en enkel kategori før det nevrale nettverket begynner å gi en akseptabel liten feilrate i tilbakemeldingene? Endrer dette antallet seg noe avhengig av hvor mange ut nevroner nettverket har? I så fall positivt eller negativt?

Spesielt med hensyn til det siste spørsmålet kan det også være interessant å teste med forskjellige nevrale nettverksstrukturer.

Jeg har etter råd fra veileder valgt å skrive denne oppgaven på norsk. Det har gjort det enklere for meg å uttrykke meg og forhåpentligvis gjort oppgaven mer lettlest og noenlunde grammatikalsk korrekt. Det største problemet med å skrive på norsk er at all vesentlig litteratur og forskning foregår på engelsk. For nær de fleste faguttrykk i denne oppgaven finnes det ikke, så vidt jeg vet, tilsvarende norske uttrykk. Jeg har etter beste evne forsøkt å oversette, men har ikke alltid lykkes. Derfor har jeg så godt jeg har kunnet inkludert og til tider brukt de engelske uttrykkene der jeg føler mine oversettelser kommer til kort.

Når det så kommer til programmering har jeg ene og alene skrevet på engelsk fordi programmeringsspråkene er på engelsk. Jeg regner med at lesere av denne oppgaven vil beherske begge språk, og jeg anser det for bare forvirrende å blande inn norsk der. Har blitt gjort oppmerksom på at det kan være mer forvirrende at jeg også har benyttet engelsk i alle oversikter over testresultater jeg har satt sammen (i Microsoft Excel). Dette skyldes at testingen har vært gjort i tilknytning til programmeringen, og ofte med direkte utskrifter fra test applikasjonene.

Forøvrig må Excel objekter konverteres til bilder for å roteres hvis noen skulle ha grunnlag til å undres over det.

1.7 Praktiske bruksområder

Det opplagte bruksområdet for et slikt system er en implementasjon som mer og mer vil opptre som et autonomt kategoriseringssystem. I en slik rolle vil det ha behov for mindre og mindre grad av korreksjon og i større og større grad avlaste brukerne av systemet for kategoriseringsarbeid.

Når et slikt system har nådd et akseptabelt nivå for gjenkjenning av objekter og/eller semantisk innhold, kan en ta det i bruk til oppgaver der det ikke kan gis tilbakemeldinger så som søking, overvåking, automatisering, og så videre.

Slik jeg ser det er det jeg foreslår i denne oppgaven kun en produksjonsmetode. Den resulterende algoritmen burde kunne ha et atskillig større bruksområde enn det relativt snevre kategoriseringsarbeidet jeg i hovedsak vil beskrive. Andre eksempler på bruksområder er generell mønstergjenkjenning, tidsserie (time series) prediksjon, diagnostisering, optimalisering, finans analyse, signal prosessering, målrettet markedsføring, og så videre. Legg merke til at dette alle er områder hvor nevrale nettverk allerede benyttes. ([39]) Av disse alternative eksemplene er dog ikke alle like enkle eller ønskelige å forene med sosiale nettsted strukturer. For eksempel forbindes gjerne ikke finansanalyse med sosiale nettsteder, men heller med nyhetstjenester.

I denne oppgaven har jeg holdt meg til ett sosialt nettsted for eksemplifisering, nemlig Flickr.com. Det betyr også at jeg begrenser meg til bilder som datagrunnlag i mine eksempler.

1.8 Fremtidig forskning

Et interessant aspekt ved Flickr er at api'et deres støtter egenproduserte tilleggsmoduler. Med api menes "Application Programming Interface" – altså at et program har et grensesnitt for aksessering. Det skulle altså være mulig å lage en slik modul med et nevralt nettverk til basis algoritme som benytter Flickr som billedbase. Videre kan en tenke seg å teste ut tags og folksonomier som grunnlag for eksperimentell opplæring av en slik maskinlæringsalgoritme.

Et aspekt ved Flickr som gjør det attraktivt i en slik setting er at api'et deres tillater å finne bilder med en gitt størrelse. Hvis en bruker retinale inndata nevroner (ett billedpunkt = en input) blir det fort en enorm mengde forbindelser og dermed utregninger som må gjøres. Som eksempel kan vi se på [30] som ved 20x20 retinale input brukte to forskjellige nettverk med henholdsvis 52 og 78 skjulte nevroner som ble til 2905 og 4357 forbindelser. Øker vi til 40x40 blir dette tilnærmet til 2905^2 = 8,5 millioner forbindelser da alle nevroner i først lag trenger forbindelse til alle retinale inndata nevronene. Jeg har etter å ha sett på gjeldende forskning valgt å fokusere på fire forskjellige måter å løse det store behovet til regnekapasitet på:

- Effektivisere nevrale nettverk. Dette kan skje ved utvikling og bruk av nye metoder som for eksempel "convolutional" nettverk og pyramidale nettverk som minker behovet til antall nevroner og forbindelser mellom disse.
  Andre løsninger kan være automatiserte forprosesseringer som fjerner

selve mønstergjenkjennelsesprosessen fra det nevrale nettverket og dermed gjøre dette til kun en beslutningsenhet. Problemet med slike løsninger har tradisjonelt vært automatiseringen – forprosesseringen har som hovedregel bestått av manuell ekspert kunnskap og tilrettelegging.

- Kraftigere maskiner som gir mer regnekraft. Dette vil høyst sannsynlig fortsette å komme i uoverskuelig fremtid og dermed tillate mer og mer komplekse nevrale nettverk. Det ser også ut til at ny teknologi vil kunne opererer på en mer egnet måte i henhold til nevrale nettverk. Mange slike teknologier har allerede blitt laget som for eksempel multikjerne prosessorer, parallelle prosessorer og DNA maskiner.
- Ta i bruk grid liknende teknologi. Det vil i prinsippet være samme løsning som punktet over, men istedenfor kraftigere enkelt maskiner lager man supermaskiner ved å kombinere mange maskiner til en virtuell maskin. Dette kan gjøres lokalt eller globalt (via internett). IBM har for eksempel laget en egen supermaskin for testing av Spiking Neural Networks.
- Teknologi for å trene opp flere nevrale nettverks agenter hver for seg for deretter å kombinere dem.

1.9 Oppgavens oppbygning

Neste kapittel vil ta for seg nevrale nettverk. Deretter vil jeg gå igjennom sosiale nettsteder med relevante kategoriseringsmodeller som tagging og taksonomi. Så vil jeg gå igjennom datamineringsteknologi som er relevant med hensyn til nevrale nettverksmodeller og da spesielt med tanke på eksportering og distribuering av slike. Jeg vil så drøfte dette med hensyn til muliggjøring av beskrevne problemstilling. Til slutt vil jeg søke å komme til en konklusjon.

Vedleggene består av kildekoden til egenprodusert applikasjoner samt bilde av brukergrensesnittene.

2 Nevrale nettverk

2.1 Hva er et nevralt nettverk?

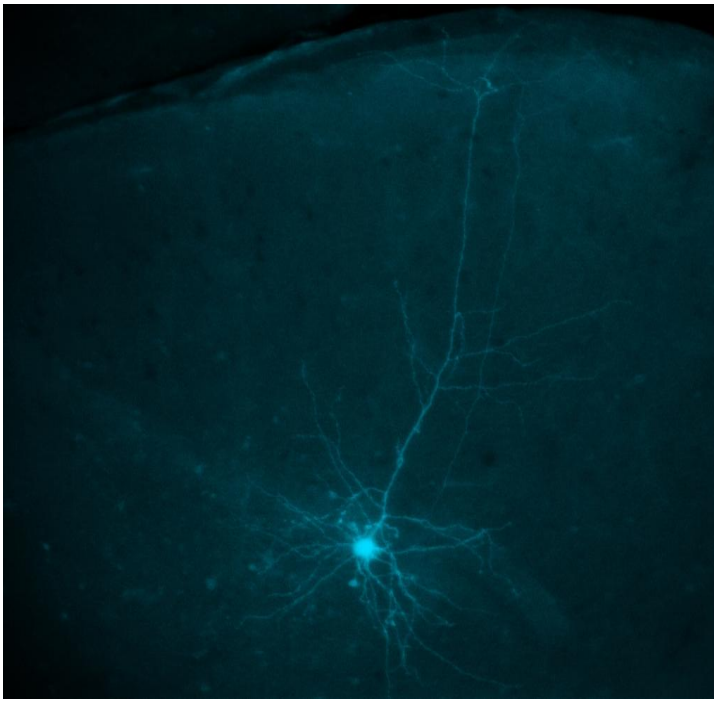
Nevrale nettverk er enheter kalt nevroner, koblet sammen i nettverk. Det er ikke uvanlig at begrepet blir både misbrukt og misforstått. Den mest basale grupperingen av nevrale nettverk går mellom kunstige og biologiske. Siden de kunstige gjerne er bygget opp med de biologiske som modell kan det være greit å starte med kort å forklare de biologiske. Det kan dog ikke understrekes nok at kunstige nevrale nettverk er noe helt annet både betingelsesmessig, kapasitetsmessig og ikke minst funksjonsmessig, enn biologiske nevrale nettverk.

2.1.1 Biologiske nevrale nettverk

Kunstige nevrale nettverk hører innunder såkalte nevrologisk inspirerte modeller, så la oss se hvordan kognitiv psykologi definerer konneksjonistiske, nevrale nettverk. [42] henviser til Rumelhart, Hinton & McClelland (1986) som de sier peker på følgende grunntrekk blant slike nettverk:

”*1. Et sett av prosesseringsenheter, kalt «noder» eller «kognitive enheter». Disse likner på nevroner, men er mye enklere og har som eneste funksjon å aktiviseres til et eller annet nivå.*

2. En aktiveringstilstand. Noder kan aktiveres i varierende grad. Dersom aktiveringsnivået overstiger en viss terskel, kobles bevisstheten inn, samtidig som andre noder er aktive utenfor bevissthetssfæren.



Figur 1: © Mark N. Miller, Nelson Lab, Brandeis University: Et pyramidalt nevron fra hjernebarken

3. Et mønster av forbindelser mellom noder. Forbindelsene er enten eksisterende (en node aktiverer en annen node), eller inhiberende (aktiveringen av en node hemmer aktiveringen av en annen node).

4. Aktiveringsregler for nodene. Disse spesifiserer hvordan en node «summerer opp» sin aktivering, hvordan inntak forbindes med eksisterende aktivering, raten for nedtrapping av aktivering, og så videre.

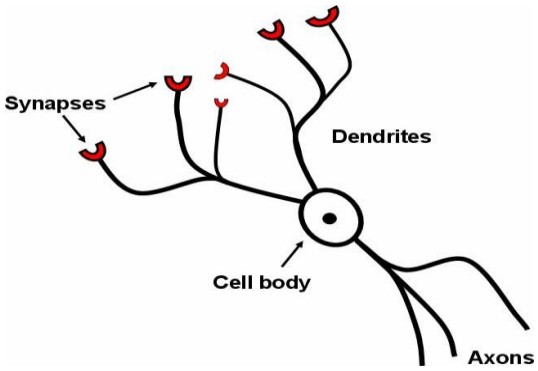
5. Uttaksfunksjoner for nodene, for eksempel hvordan uttak skal relateres til aktuell aktivering.

6. Regler for læring. Læringen består i å styrke forbindelsene mellom nodene, og dette kan foreskrives av ulike typer regler. Den enkleste er såkalte Hebb-regelen, som slår fast at forbindelsen mellom to noder styrkes når de aktiveres samtidig.

7. Et «miljø» for systemet. Dette vil si at noder går sammen i moduler som har spesifikke funksjonsoppgaver (som for eksempel å gjenkjenne ansikter). Ulike moduler antas å være massivt koplet med andre modulenheter."

Videre sier [42]: "*Det er svært viktig å være klar over at det ikke er noen kunnskap* inne i *de enkelte nodene. Kunnskapen ligger i mønsteret og styrken av forbindelser i nettverket.*"

Figur 1 viser et ekte pyramidalt nevron med dendritter (forbindelser), altså en hjernecelle. Figur 2 viser en skjematisk fremstilling av det samme, dog såpass simplifisert at å kalle det en fremstilling av en pyramidal celle vil være å trekke det langt. Et typisk nevron fra hjernebarken til et menneske har rundt 10'000 dendritter som typisk strekker seg utover i alle retninger.



Figur 2: © J. Zupan & J. Gasteiger 1999

Med unntak av punkt 7 og inhibering i punkt 3 er alle disse punktene også gjeldende for og til sammen en god definisjon på kunstige nevrale nettverk. Men det finnes kunstige nevrale nettverksmodeller som benytter hele punkt tre. Videre er [43] et eksempel på bruk av punkt 7, dog kun til eksperimentelle formål og ikke intensjonelt. Snarere var det en av de interessante observasjonene gjort av [43] at det kunstige nevrale nettverket tenderte til å dele seg opp i grupperinger helt av seg selv.

Forskjeller mellom biologiske nevrale nettverk og kunstige nevrale nettverk er mange og fundamentale. For eksempel:

- De biologiske skapes med predefinerte strukturer og mange ferdig, fungerende funksjoner. I et kunstig nevralt nettverk initialiseres normalt

alle vekter til vilkårlige verdier og har ingen funksjonalitet overhodet før dette spesifikt utvikles.

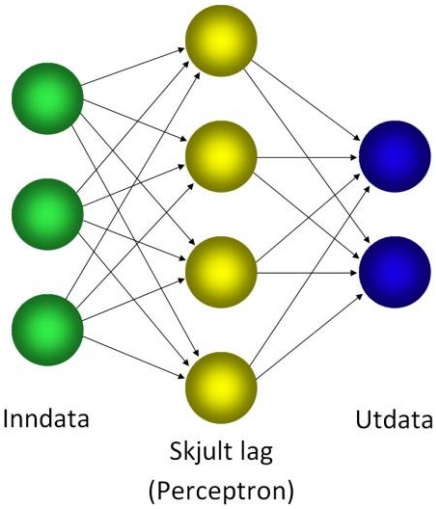
- Nevronene i kunstige nevrale nettverk er ekstremt simplifiserte, gjerne ikke mer enn en summeringsformel med et sigmoid filter for å gi en tilpasset utverdi. Et nevron i et biologisk nevralt nettverk derimot er en - selvfølgelig biologisk - celle som kan utføre funksjoner avhengig av hva som kommer som inndata.
- Biologiske celler opererer med firetallssystem (DNA) mot binært tallsystem som benyttes av elektroniske datamaskiner.
- Dendrittene kommuniserer med de biologiske nevronene ved hjelp av flere forskjellige kjemikalier på avanserte måter vi enda ikke har full oversikt over. Koblinger mellom nevroner i kunstige nevrale nettverk består i å multiplisere ett tall med en vekt.
- Koblinger mellom nevroner i et biologisk nevralt nettverk opprettes ved behov. De vanligste kunstige nevrale nettverksmodellene opererer med fastsatt struktur, men det finnes altså varianter som oppretter koblinger etter behov. Men ingen av de mest avanserte variantene for objektgjenkjenning i for eksempel billedmateriale, slik som CNN og Pyranet, gjør dette per i dag.

Det er en pågående forskning for å lage kunstige nevrale nettverksmodeller som kan utføre mer og mer lik simulering av biologiske nevrale nettverksmodeller. Men som [43] viser, er moderne forskning knapt nok i nærheten av å kunne utføre minimaliserte simuleringer av kapasitetsmessige aspekt ved biologiske nevrale nettverk.

([42], [140])

2.1.2 Kunstige nevrale nettverk

Hvorfor ønsker man så å lage et kunstig nevralt nettverk? I følge [5] lager man maskin lærings algoritmer for ikke bare å få en maskin til å produsere korrekte utdata, men for at maskinen skal produsere disse dataene lik en menneskelig kognitiv prosess. I lys av dette er foreslått opplæringsmetode helt korrekt. For grunnlaget til det nevrale nettverket i vårt tilfelle er en samling av hva mennesker oppfatter et objekt til å være eller inneholde. Det som det nevrale nettverket kalibreres til å gi som utdata for gitte inndata samsvarer med resultatet til en menneskelig kognitiv prosess. Egenskapene



Figur 3: Et nevralt nettverk med tre inn nevroner, fire skjulte nevroner og to ut nevroner med full kobling i mellom lagene.

til det kunstige nevrale nettverket må derimot altså ikke forveksles med egenskapene til biologiske nevrale nettverk.

2.1.3 Kunstige nevrale nettverk kontra andre maskinlærings algoritmer og kunstig intelligens algoritmer

Kunstige nevrale nettverk har til nå og er fremdeles gjerne ansett som sorte bokser. Med det menes at en presenterer inndata for algoritmen og får ut et resultatsett uten å vite hvordan algoritmen har kommet frem til dette resultatet. Andre maskinlæringsalgoritmer som grupperingsalgoritmer med utgangspunkt i statistikk samt kunstig intelligens algoritmer, er derimot spesifisert med eksakte regler. Altså kan hvert eneste steg i de spesifiserte algoritmene regnes ut og etterprøves for seg. ([5],[122])

I denne sammenheng må det også nevnes at kunstige nevrale nettverk ikke lenger er rene sort bokser. For eksempel viser LeCun et. al at man kan visuelt fremstille de forskjellige mønstrene som ekstraheres av CNN. CNN forklarer jeg nærmere i 2.4.4. ([15],[41])

2.2 Grunnleggende om strukturen til kunstig nevral nettverk

Kunstige nevrale nettverk ble først beskrevet i 1943 av Warren McCulloch og Walter Pitts. De klarte å demonstrere at slike nettverk kan utføre aritmetiske og logiske operasjoner.

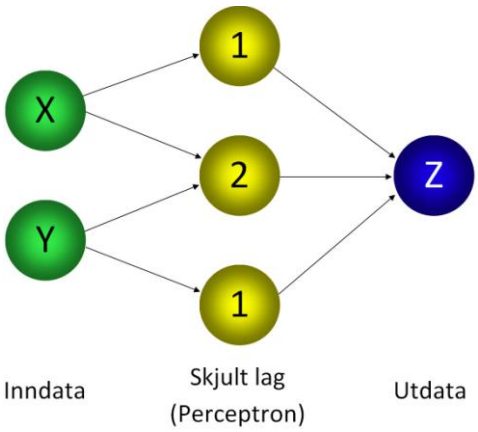
Figur 3 viser en enkel form for et nevralt nettverk av typen forovermatet (Feed forward Neural Network - FNN). Det består altså av tre nevroner som hver har en kobling til hver ev de fire nøytronene i neste lag som igjen alle har hver sin kobling til de to nevronene i utlaget. [65] kaller koblingene for dendritter slik en gjør i biologien, men det vanlige innen kunstige nevrale nettverk er å kalle dem koblinger. Noen kaller også koblinger for vekter, men dette kan være misvisende da nevronene også kan ha vekter i seg selv. Nevroner kalles også for prosesserings enheter eller bare enheter. Første og siste lag kalles gjerne for synlige enheter, mens lagene imellom gjerne kalles skjulte enheter.

Nettverket fungerer ved at inndata nevronene sender data til nevroner i det skjulte laget via veide koblinger. At koblingene er veide betyr at dataene blir multiplisert opp med den enkelte koblingens lagrede vekt før dataene blir sendt til mottaker nevronet i neste lag. Det mottagende nevronet sin funksjon er å samle sammen alle data fra innkommende koblinger, normalt ved en summeringsformel. Deretter er det vanlig å la nevronet normalisere dataene, gjerne til en verdi mellom 0 og 1 ved hjelp av en sigmoid funksjon. Normalt sendes så resultatet videre til alle nevronets (veide) utkoblinger, men man kan

også la nevronet ha en terskel verdi som må passeres for at nevronet skal sende data videre. Det kan godt være mer enn et lag med skjulte nevroner.

Til slutt kommer det data til utnevronene. Her er det vanlig å ha en terskel verdi slik at ut nevronene leverer enten 0 eller 1 – sant eller ikke sant – fant eller fant ikke – osv. Men en av styrkene til nevrale nettverk er at de kan behandle data i en tilnærmet analog form; utdataene kan godt være et tall *mellom* 0 og 1, og dermed angi en sjanse for funn/ikke funn fra inndataene fremfor sant/ikke sant.

Figur 4 viser et kunstig nevralt nettverk som er i stand til å løse XOR problemet. Sett inn verdier for x og y, multipliser med tallene ved siden av vektene og benytt tallene i de skjulte nevronene som grense verdier. Er tallet som kommer til et skjult nevron større eller lik grenseverdien sendes tallet 1 videre.



Figur 4: z = XOR (x, y), pilene er koblinger, rundingene nevroner. ([141])

Det kan også varieres med antall utparametre og eventuell samhandling disse imellom. En vanlig bruk er inhiberende koblinger når en ønsker å la kun et av utdata nevronene gi et svar, for eksempel ved klassifisering. Man lager da koblinger mellom alle ut nevronene, og den som har fått høyest inndata ”vinner” og blir det eneste nevronet som sender noe videre. Legg merke til at ved slike koblinger mellom ut nevronen er det nevrale nettverket ikke lenger et FNN siden det finnes koblinger innad i et og samme lag.

I figur 3 og 4 er alle nevroner i et lag knyttet til alle nevroner i neste lag. Det trenger ikke være tilfelle. Det kan for eksempel godt være en begrensning på antall koblinger. Figuren beskriver det en vil si er et endimensjonalt nettverk. Ved et todimensjonalt nettverk er nevroner i et lag organisert (skjematisk sett) langs to akser. I et slikt nettverk har en muligheten til å ta et utsnitt – gjerne rektangulært – av nevroner og lage koblinger til et utsnitt av nevroner i neste lag. Det kan godt være overlapping mellom de forskjellige utsnittene. Poenget er gjerne å redusere antall koblinger *og* å redusere antall nevroner i neste lag, og dette gjerne for subsampling.

En annen og mer avansert form for nevrale nettverk er sykliske nevrale nettverk (RN – Recurrent neural Network). Lages for eksempel koblinger fra utdata nevroner til inndata nevroner har man en slik variant. Den mest ”komplekse” form for sykliske nevrale nettverk er fullkoblede nevrale nettverk hvor alle nevroner er koblet til alle. Slike nettverk kalles Hopfield nettverk.

Sykliske nettverk stiller selvfølgelig andre krav til læringen i forhold til forovermatede nettverk, og da især Hopfield type nettverk.

Det er gitt at et rent forovermatet nevralt nettverk som beskrevet over og som mates med samme data om og om igjen, vil konvergere mot en stabil (fast) tilstand. Hvis dette ikke er ønskelig kan et stokastisk (tilfeldig, ikke entydig - men tilnærmelsesvis beregnbar statistisk) element legges til i nevronene. En nettverksform som gjør dette kalles en Boltzmann Machine. ([66])

2.3 Grunnleggende om trening av nevrale nettverk

En av de enkleste og mest brukte teknikker for å lære opp et forovermatet nevralt nettverk er bakover propagering (backpropagation learning). Teknikken ble først beskrevet av Paul Werbos i 1974 ([9], [13], [51]). Prinsippet er enkelt og går ut på at når utdata nevroner leverer et resultat sjekkes dette mot et fasitsvar. Er svaret til et nevron feil sender det en melding tilbake til sine (inn-) koblinger slik at de korrigeres. Koblingene sender så beskjeden videre til sine inn-nevroner om at svaret var feil. Dermed vil disse gjenta korrigeringen videre bakover i nettverket. Altså; når en gir et utnevron beskjed om å korrigere sitt svar vil korrigeringer bli gjort i alle bakenforliggende vekter – ergo bakover propagering. Videre; hvis svaret var korrekt, brukes bakover propagering til å styrke vekter som var involvert i å produsere det korrekte svaret.

Enkleste form for optimalisering av bakover propagert læring er å vekte ”skyld” for riktige eller feil svar og variere korrigeringen av vekter med selvfølgelig størst korreksjon på de vekter som hadde mest skyld og mist til de med minst skyld.

Videre optimaliseres det for hvor store endringer som skal gjøres på vektene ved hver iterasjon. Hva som er beste justering kan avhenge av størrelse til nettverket, type nettverk og type data som skal behandles. Det finnes altså ingen fasit på hvor store endringer som bør gjøres for hver iterasjon.

Lippman (1987) skal være en akseptert standard for generell utforming av bakover propagerings algoritmer og går som følger:

Gjenta

velg et datasett som inndata;

regn ut ønsket resultat for det nevrale nettverket;

juster vekter og terskler;

til stopp kriterium er oppnådd.

([56])

La oss tenke oss et objekt orientert utgave av et nevralt nettverk hvor altså alle funksjoner for justering av hvert objekt ligger i og blir utført av hvert enkelt objekt. La oss videre si at en av funksjonene til nevron objektene er en bakover propagerings funksjon. Introduseres det da en kobling som går bakover i det nevrale nettverket vil det oppstå en uendelig løkke. Med mindre det finnes en form for impuls kontroll eller avbrytningsfunksjon for nettverket vil en iterasjon aldri avslutte og nye inndata vil ikke kunne mates inn.

Tilsynelatende enkle endringer i nettverksstrukturen kan altså gi drastiske endringer i virkemåte og vil betyr helt andre forutsetninger for læringsteknikk.

Det betyr ikke at sykler nødvendigvis skal unngås; en mulig virkemåte for et syklisk nevralt nettverk som benytter bakover propagering er hukommelse. Denne hukommelsen vil gjerne være av typen korttidsminne når benyttet innen kunstige nevrale nettverk, og kan for eksempel brukes til evaluering av serier.

En trenger altså ulike typer læringsalgoritmer for ulike nevrale nettverksformer og for ulike bruksområder. Forskjellige typer læringsteknikker kan også variere kraftig i effektivitet – som regel i direkte sammenheng med krav til maskinvare. Skal gå nærmere inn på dette senere.

([7], [8], [16], [18], [64])

2.3.1 Varianter av læringsteknikker

Læringsteknikker for nevrale nettverk deles inn i tre kategorier:

- Kontrollert læring (supervised learning)
- Ukontrollert opplæring (unsupervised learning)
- Læring ved forsterkning (reinforced learning)

Ved kontrollert læring har man sett med både inn og ut data. En læringsalgoritme justerer det nevrale nettverket til det leverer korrekte resultater i henhold til inndata. Nevnte bakover propagerings algoritmer er et eksempel på en slik kontrollert læringsteknikk.

Ved ukontrollert læring av det nevrale nettverket stiller man uten apriori klassifisering av data. I stedet går teknikken ut på å la en algoritme se på inndata og justere det nevrale nettverkets utdata etter gitte kriterier for dataene. Slike kriterier kan for eksempel være clustering (gruppering) eller gjennomsnittsverdi. SOM (Self-Organizing Map) også kalt SOFM (Self-Organizing Feature Map) og Kohnen Map, er en nevral nettverksmodell som er laget for slik form for læring.

Forenklet forklart er SOM en omvendt utgave av gruppering hvor en istedenfor å finne senter og utstrekning av grupperinger av data, regner seg frem til vektorer som gir størst mulig avstand mellom grupperinger.

Uhøytidelig forklart er læring ved forsterkning en hermeteknikk. Denne brukes når det ikke finnes apriori data. Istedenfor settes det nevrale nettverket til å herme etter inndata. Det vil si - teknikken går ut på å forsøke å minimalisere avvik fra fremtidige data basert på akkumulert inndata. Så i tillegg til ukontrollert læring har læring ved forsterkning en kontroll funksjon.

MDP (Markov decision process) er et matematisk rammeverk som ofte benyttes for å forutse fremtidige forekomster og er et eksempel en teknikk som benyttes til læring ved forsterkning.

([7], [8], [16], [18], [64], [122])

2.4 Varianter av nevrale nettverk

Det finnes nær sagt utallige variasjoner av nevrale nettverk. Jeg vil gå nærmere igjennom noen av de jeg anser som viktigst i henhold til denne oppgave.

2.4.1 MLP – Multi-Layer Perceptrons

MLP er ifølge [2] den vanligste form for nevrale nettverk per i dag. Rett og slett et vanlig forovermatet nevralt nettverk slik beskrevet over, men med mer enn et skjult lag med nevroner.

Det er vist at det for MNist datasettet (se 2.2 eksempler på bruk; OCR) her det blitt oppnådd en feilrate på helt ned til 0,6%. Til sammenlikning er det beste resultatet som hittil har blitt oppnådd på MNist 0,39% feilrate.

Selv om [40] sier at det ikke finnes noen regel om hvor mange skjulte lag med nevroner som bør være i et MLP sier de samtidig at hvert skjulte lag representerer et abstraksjonsnivå. Det er forøvrig bevist at også menneskers nevrale nettverk jobber i forskjellige abstraksjonsnivåer. ([42], [140])

2.4.2 RNN eller bare RN – Recurrent neural Network

I motsetning til MLP'er går informasjonen ikke bare forover i det nevrale nettverket. RNN er altså nevrale nettverk med sykler. Som allerede nevnt betyr det andre forutsetninger for hvilke læringsteknikker som kan benyttes.

Simple Recurrent Network (SRN) – også kalt Elman nettverk etter oppfinneren Jeff Elman - ble introdusert i 1990 og er et eksempel på RNN. Det benytter en tre lags struktur og legger på et ekstra sett med nevroner i det første laget. Koblinger med en vekt på 1 går fra det skjulte laget tilbake til dette ekstra settet med nevroner. Altså vil det ekstra settet bestå av like mange nevroner som i det skjulte laget. Dermed beholdes en kopi av forrige sett med inndata og nettverket kan utføre oppgaver som for eksempel sekvens prediksjon.

2.4.3 Kohnen feature maps (SOM)

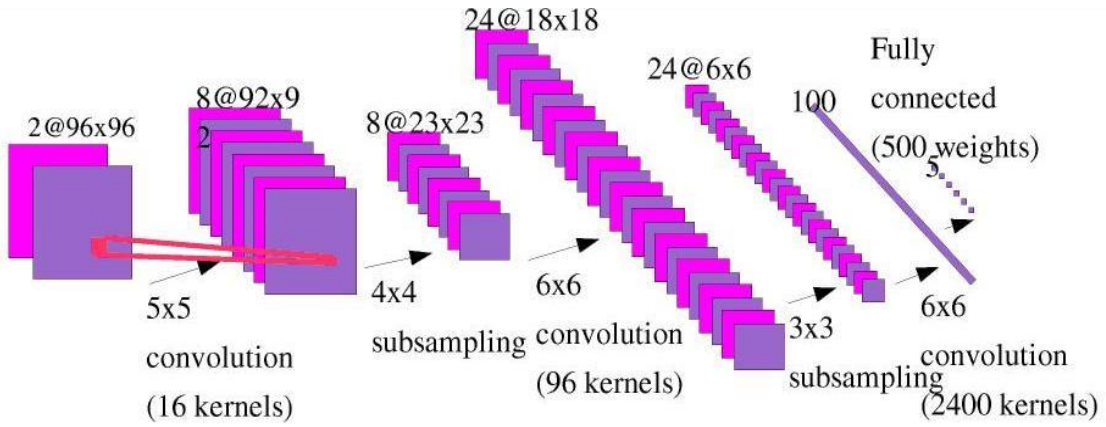
Oppfunnet av Teuvo Kohonen, derav navnet. Også kalt selvorganiserende egenskaps kart - Self Organizing feature maps (SOM), eventuelt (SOFM). Et eksempel på en type nevralt nettverk som benytter ukontrollert læring (unsupervised learning). Det vil si at nettverket ikke fortelles hva som er riktig eller galt svar på inndata. Slike nevrale nettverk kan benyttes til finne strukturer som for eksempel grupperinger.

Det benyttes multidimensjonelle inn- og utdata, og antall dimensjoner i inndataene og i utdataene trenger ikke å være det samme. Som eksempel kan man altså bruke et SOM til nedskalering av 3D til 2D.

Inspirasjonen til SOM er delvis hentet fra hvordan forskjellige deler i den menneskelige hjernen behandler data fra sanser som blant annet syn og hørsel. I henhold til Kohnen er SOM den type kunstig nevralt nettverk som er nærmest å simulere faktisk virkemåte til biologiske (hjerne) nevrale nettverk.

[69] er et eksempel på praktisk bruk av SOM hvor det brukes til å fjerne motstridende data.

2.4.4 CNN - Convolutional Neural Network – Intrikat, Foldbart Nevralt Nettverk



Figur 5: © Yann LeCun (2007) - Architecture of the convolutional net "LeNet 7". This network has 90,857 trainable parameters and 4.66 Million connections. Each output is influenced by a receptive field of 96x96 pixels on the input.

om man vil, bestemte former. Mellom en del av disse form kartleggingslagene benytter man lag for nedskalering (subsampling) av data. Det resulterende nevrale nettverket minsker behovet for antall nevroner og forbindelser både grunnet nedskaleringen, men først og fremst at mønstre lagres ideelt sett én gang i nettverket uavhengig av hvor det befinner seg i inndataene. CNN har også vist seg å være mer robuste med hensyn til støy og transformasjoner. Transformasjoner kan for eksempel i billeddata være rotasjon, vridning, med mer.

Professor Yann LeCun som er en av oppfinnerne av CNN, jobber sammen med andre med å benytte modellen til generisk objektgjenkjenning i bilder. De vil si at de forsøker å få et CNN til å gjenkjenne objekter direkte fra et piksel basert bilde utavhengig av lysforhold, bakgrunn, positur og støy. ([41], [7], [3])

2.4.5 TDNN – Time-Delay Neural Network

Introdusert av Alex Waibel. En variasjon av CNN som, som navnet tilsier, tar hensyn til tid i inndata. Brukes blant annet ved analyse av video og lyd. [14] har benyttet TDNN til fonem gjenkjenning (atomære deler av tale). Strukturen til algoritmen består av et trelags forovermatet nevralt nettverk med forsinkelsesenheter før hver vekt, altså før koblingene til nevronene. Til læring benyttes bakover propagering.

2.4.6 PyraNet - Pyramidal Neural Network

En av de nyeste nevrale nettverksmodellene designet spesifikt for mønster gjenkjenning i bilder er PyraNet. [1] gir et fullt matematisk regelsett for hvordan et slikt nevralt nettverk vil se ut og hvor stort det blir. Samtidig gir de utfyllende opptegnelser på diverse ytelsesmålinger samt sammenlikninger med andre relevante nevrale nettverksmodeller. Sammenlikningen med CNN er dog mot LeNet5 og et annet CNN nettverk med omtrent samme ytelse som LeNet5 (0,1% forskjell i klassifiseringsrate). Ranzato et al. 2006 sin LeNet6 løsning med benyttelse av ukontrollert læringsteknikk (unsupervised learning) kan halvere feilraten til LeNet5 og burde dermed kunne komme opp i omtrent samme klassifiseringsrate som Pyranet. Grunnet full tilgang på det matematiske grunnlaget, den relativt enkle oppbygningen og den, relativt til andre modeller, høye ytelsen, vil jeg se litt mer detaljert på PyraNet i forhold til andre nevrale nettverksmodeller jeg ser på. ([1], [7])

PyraNet benytter en strukturert subsampling med overlapp. Målet er det samme som ved CNN; å unngå multippel lagring av et og samme mønster (feature) i det nevrale nettverket, samt å minimalisere antall trenbare vekter. Med subsampling menes å sette koblinger mellom et gitt antall nevroner til et og kun et nevron i neste lag. Setter en dette til for eksempel å være 16 nevroner koblet til et nevron i neste lag, får en altså senket antallet nevroner fra et lag til neste med 1/16. Hvis en tenker på hvert nevronlag som todimensjonalt (liggende flatt) og med hver subsampling som et rektangulært utsnitt, vil altså en opptegning få en pyramide form. Å gjøre subsamplingen i 2D er selvfølgelig ikke tilfeldig med tanke på at det er 2D bilder som skal behandles.

La oss ta en titt på hvilke krav dette stiller i praksis. [1] har satt opp nøyaktige matematiske beskrivelser av antall parametere deres modell resulterer i. I Tabell 1 har jeg tatt utgangspunkt i de matematiske beskrivelsene og lagt til utregningsdata for bilder med relativt høy oppløsning, nemlig 500x500 og 2048x2048. Jeg har antatt verdier for r (bredde på utsnitt) kan økes noe for større nettverk og at o (overlapp mellom utsnitt) kan justeres. Jeg har så valgt verdier for r og o som får høyde og bredde i neste lag til å gå opp, altså ikke få desimaler på høyden eller bredden noe sted.

		PyraNet størrelse							
Størrelse bilde	**Høyde (H0)**	**20**	**26**	**38**	**50**	**500**	**500**	**500**	**2 048**
	Bredde (W0)	**20**	**22**	**38**	**42**	**500**	**500**	**500**	**2 048**
	Antall nevroner (N0)	400	572	1 444	2 100	250 000	250 000	250 000	4 194 304
Størrelse pyramide lag 1	Reseptor bredde (R1)	5	4	4	4	4	4	4	5
	Overlapp (O1)	2	2	2	2	2	2	2	2
	Høyde (H1)	6	12	18	24	249	249	249	682
	Bredde (W1)	6	10	18	20	249	249	249	682
	Antall nevroner (N1)	36	120	324	480	62 001	62 001	62 001	465 124
Størrelse pyramidelag 2	Reseptor bredde (R2)	4	4	4	4	5	5	5	4
	Overlapp (O2)	2	2	2	2	3	3	3	2
	Høyde (H2)	2	5	8	11	123	123	123	340
	Bredde (W2)	2	4	8	9	123	123	123	340
	Antall nevroner (N2)	4	20	64	99	15 129	15 129	15 129	115 600
Størrelse pyramidelag 3	Reseptor bredde (R3)			4		5	5	5	4
	Overlapp (O3)			2		3	3	3	2
	Høyde (H3)			3		60	60	60	169
	Bredde (W3)			3		60	60	60	169
	Antall nevroner (N3)			9		3 600	3 600	3 600	28 561
Størrelse pyramidelag 4	Reseptor bredde (R4)					4	4	4	5
	Overlapp (O4)					2	2	2	3
	Høyde (H4)					29	29	29	83
	Bredde (W4)					29	29	29	83
	Antall nevroner (N4)					841	841	841	6 889
Størrelse pyramidelag 5	Reseptor bredde (R5)					5	5	5	5
	Overlapp (O5)					2	2	2	2
	Høyde (H5)					9	9	9	27
	Bredde (W5)					9	9	9	27
	Antall nevroner (N5)					81	81	81	729
Beslutningsnettverk lag 1		1	1	1	1	1	25	50	100
Beslutningsnettverk lag 2							1	25	25
Totalsum nevroner og vekter (P)		**481**	**853**	**2 239**	**3 259**	**413 305**	**415 299**	**418 598**	**5 502 906**
N0 som andel av P		83,2 %	67,1 %	64,5 %	64,4 %	60,5 %	60,2 %	59,7 %	76,2 %
N0+N1 som andel av P		90,6 %	81,1 %	79,0 %	79,2 %	75,5 %	75,1 %	74,5 %	84,7 %

Tabell 1: Egne utregninger for "store" nevrale nettverk basert på tabell I fra [1].

Årsaken til at jeg har valgt 500x500 piksler som størrelse er at 500 er maksimal verdi for bredde eller høyde på et standard visningsbilde i Flickr.com. Ser at allerede ved en slik relativt liten billedstørrelse er det subsamplingen med den tilhørende mønstergjenkjenning (2D pyramide nettverket) som opptar mesteparten av samlet antall nevroner og vekter. Selv økning fra 1 til 25 ut parametere har liten betydning (8‰ fra kolonne 5 til kolonne 6) på P (totalt antall nevroner og vekter).

Økning fra 1 til 25 utparametre, altså tilgjengelig antall klassifiseringer, vil derimot ikke være irrelevant for treningen. [1]

Videre kan vi se at R1 (sammen med O1) har stor innvirkning på P. Faktisk står N0 og N1 for 74,5% til 84,7% av alle trenbare parametere i nevnte eksempler.

Hvordan gir så dette seg utslag i det virkelige liv?

Minne bruk ved trening

Trenings Metodikk	Utregnings grunnlag	Billedstørrelse 20x20 K=14400	26x22 K=14400	38x38 K=14400	50x42 K=14400	500x500 (1) K=10000	500x500 (2) K=10000	500x500 (3) K=10000	2048x2048 K=10000
GD	K x P	6,6M	11,7M	30,8M	44,8M	3,8G	3,9G	3,9G	51,2G
GDMV	K x P	6,6M	11,7M	30,8M	44,8M	3,8G	3,9G	3,9G	51,2G
RPROP	(K + 2) x P	6,6M	11,7M	30,8M	44,8M	3,8G	3,9G	3,9G	51,3G
CG	(K + 3) x P	6,6M	11,7M	30,8M	44,8M	3,9G	3,9G	3,9G	51,3G
LM	$K \times P \times N_L + P^2$	20,0M	35,8M	127,8M	144,4M	182,2G	187,7G	190,5G	27,9T

Tabell 2: Egne minnebruk-beregninger gjort på grunnlag av tabell IV fra [1] sine matematiske modell. Her beregnet med forholdsvis store datasett (K=10000 til 14400).

I Tabell 2 har jeg brukt 2 som grunntall for omregning til M, G og T. Altså M = 1'048'576. Videre benytter alle eksempler i [1] gråtoner, så 1 i denne sammenheng kan sammenstilles med 1 byte som tilsvarer 1 piksel. Forøvrig vil ikke nødvendigvis farger tredoble størrelse. Farger kan sees på som tre til fire dimensjoner som for eksempel rød, grønn og blå eller cyan, gul, magenta og intensitet. Men disse dimensjonene kan redimensjoneres til 2D ved hjelp av for eksempel Eigenvektorer. Kort sagt; farger er et kapittel for seg.

Tilbake til resultatene ser vi at vi med 500x500 billedpunkter i gråtoner kan komme i overkant av det en "normal" moderne arbeidsstasjon hvis læringsalgoritmen settes opp slik [1] har gjort. Per i dag er kommer prekonfigurerte arbeidsstasjoner normalt med 1GB til 2GB minne ([17]). Avhengig av type operativsystem må det normalt trekkes fra 200 til 500MB. Regner 800MB som gjengs ledig tilgjengelig minne. Heldigvis er resultatene over basert på at hele treningsgrunnlaget må være tilgjengelig for treningsalgoritmen. Dette stemmer fullt ut kun for de aller raskeste treningsteknikkene, i dette tilfellet LM. Til [1] sitt forsøk hadde de manuelt samlet sammen 20'000 bilder hvorav 14400 ble brukt til opplæring og resterende til ytelsesmåling. I andre sammenhenger vil eller må en kanskje benytte færre bilder per treningsrunde. Setter vi antall bilder tilgjengelig for treningsalgoritmen til 500 ad gangen får vi tall som i Tabell 3.

Minne bruk ved trening, K = 500

Trenings Metodikk	Billedstørrelse 20x20	26x22	38x38	50x42	100x100	250x250	500x500	2048x2048
RPROP	235,8K	418,2K	1,1M	1,6M	7,7M	50,0M	200,4M	2,6G
LM	930,5K	1,9M	9,1M	14,8M	293,0M	10,5G	164,6G	27,6T

Tabell 3: Egne minnebruk-beregninger gjort på grunnlag av tabell IV fra [1] sine matematiske modell. Her beregnet med forholdsvis lite datasett (K=500).

Med dette utgangspunktet kan en slik modell benyttes med god margin helt opp til 500x500 billedpunkter på alle nevnte læringsalgoritmer med unntak av LM.

Testene viser at 26x22 billedpunkter i de data som ble brukt gir høyest (best) klassifikasjonsrate. Både ved bruk av færre eller flere billedpunkter synker nøyaktigheten. Det kan altså hende at det ikke vil være lønnsomt å gå for flere billedpunkter, men dette er vanskelig å si da [1] ikke gjør forsøk med mer enn

ett utparameter. [1] sin undersøkelse er gjenkjennelse av kjønn basert på bilder av ansikter som naturlig nok fordrer kun et utparameter.

De har heller ikke tatt med grafer for utvikling av klassifikasjonsrate for forskjellige billedstørrelser satt opp mot trenings-mengde av respektive nettverk. Derimot legges det frem data som viser at de forskjellige læringsmetodene tangerer mot noenlunde samme ytelse for ett og samme nettverk hvis nok læring blir gitt. GD og GDMV tangerte her mot ca 1% dårligere klassifikasjonsrate enn de andre tre. Men de mest minnekrevende teknikkene tangerer atskillig fortere.

20x20 bilde - andre relevante data			
Trenings Metodikk	P4 2.8Ghz - 1GB mem		
	Tid (gdeu*)	**tid minutter**	**Klassifiserings rate**
GD	26 941,5	1 594,0	96,5 %
GDMV	20 726,2	1 226,3	96,6 %
RPROP	1 035,5	61,3	97,3 %
CG	735,6	43,5	97,2 %
LM	65,5	3,9	97,5 %

** Ved angitt spesifikasjon er 1 gdeu ca. 3,55s*

Tabell 4: Tid satt opp mot klassifiseringsrate med tall hentet fra [1].

Tabell 4 viser altså tidsbruk og ytelse for forskjellige teknikker for et bilde med 400 billedpunkter. I disse tilfellene ble treningen foretatt med 14400 bilder.

Det påpekes forøvrig at for eksempel forsøk med MPL har gitt et høyere toppunkt med hensyn til antall billedpunkter for best mulig klassifiseringsrate. Resultatene er altså avhengig av både nettverksmodell og treningsteknikk.

Forkortelsene som er brukt for treningsmetodikker står for følgende:

- GD: Gradient Descent [18]
  Vekter oppdateres langs den negative gradienten
- GDMV: Gradient Descent with Momentum and Variable learning rate
  Vektene oppdateres etter en lineær kombinasjon av gradienten og forrige vekt oppdatering
- RPROP: Resilient backPROPagation
  Vekt oppdatering er avhengig kun av fortegnet til gradienten
- CG: Conjugate Gradient
  Vekter oppdateres langs retningene som samsvarer med en Hessian matrise
- LM: Levenberg-Marquardt [18]
  Andre ordens Taylor utvidelse og Gauss-Newton tilnærming til Hessian matrise

2.4.7 Andre typer

Boltzmann machine neural network ([66], [67]); en stokastisk variasjon av Hopfield nettverk. Hovedproblemet med slike nevrale nettverk er at tiden det tar for å oppnå en stabil tilstand (altså læring til et nivå hvor nettverket kan brukes) øker eksponentielt størrelsen til nettverket. Dermed blir slike nettverk så fort uhåndterlige at de ikke er praktisk brukbare til komplekse problemstillinger. Det er laget en variant av Boltzmann maskin kalt RBM (Restricted Boltzmann machine) hvor det ikke tillates opprettelse av kontakter mellom nevroner i de skjulte lagene. [67] er et eksempel på at RBM kan brukes til å løse praktiske problemer.

SNN (Spiking Neural Network) ([121]) er en form for nevrale nettverk som ikke benytter epochs (et lag med nevroner kalkuleres om gangen) men isteden lar hvert enkelt nevron bestemme når det skal kalkuleres og sende utdata til de nevronene den har kobling til. Måten dette gjøres på er å implementere et aktiverings nivå i hvert enkelt nevron som bygges opp av inndata fra andre nevroner. Alle biologiske nevrale nettverk fungerer så vidt man vet på denne måten, og SNN er direkte inspirert av biologisk forskning. [65] og [43] bygger begge på SNN. [65] er et patent på en metode for mønster gjenkjenning, mens [43] er et berømt musehjerne simulerings eksperiment utført av forskere fra IBM (som tidligere nevnt i bunn og grunn ikke har noe som helst med en musehjerne å gjøre).

2.5 Eksempler på bruk

Nevrale nettverk brukes til så mangt. Jeg har her sett nærmere på hvordan de artiklene jeg valgt å benytte som grunnlag for denne oppgaven, benytter dem.

2.5.1 Ansiktsdeteksjon

[31] er en oversikt over mer enn 150 artikler som omhandlende ansiktsdeteksjon, vel og merke; ikke bare nevrale nettverksbaserte. [1],[29], [30], [46] og [67] er derimot alle artikler som beskriver metoder som benytter nevrale nettverk til å lete etter ansikt i bilder. Variasjoner ligger først og fremst i fremgangsmåte; altså om metoden forutsetter forbehandling, hvilken struktur som benyttes for det nevrale nettverket, og hva slags læringsteknikk som benyttes. Det er også variasjoner i om det lokaliseres et eller flere ansikter og om eventuelle ansiktsfunns plassering i bilde rapporteres.

Det er gjerne vanskelig å sammenlikne de forskjellige fremsatte metodene fordi forutsetningene ofte er forskjellige. Spesielt går dette på hvilket materiale som benyttes som grunnlag for læringen. Hvis dette ikke er identisk – for eksempel samme sett med bilder, vil sammenlikning gjerne være misvisende eller til og med meningsløst. Konklusjonen til [31] er nettopp en utfordring til de som

lager slike metoder å finne aksept for en generell ytelsesmåling av slike metoder. Som de sier; dette vil ikke bare gjøre det enklere for folk som søker å finne best mulig metode for å løse en gitt problemstilling, men vil også virke oppfordrende for utviklere av slike modeller til å øke effektiviteten til sine modeller.

Ansiktsdeteksjon er spesielt interessant fordi det vedrører et bassalt problem innen gjenkjenning ved hjelp av datamaskiner; nemlig et for oss mennesker klart definert objekt som har store variasjoner innen form og farge. Ikke bare kan ansikter i seg selv ha forskjellige former og farger som hudfarge, hårfarge, hårfasong, skjeggvekst, ansiktsproposjoner, og så videre. Bilder av ansikter kan også bestå av briller, sminke, og andre variabler som egentlig ikke tilhører ansiktsobjektet. Men selv med alle disse variasjonene er problemstillingen til slike algoritmer enkel; enten inneholder bildet ansikt eller så gjør det ikke det.

2.5.2 OCR – Optical Character Recognition

Gjenkjenning av tekst er en egen industri som daterer helt tilbake til 1929 da Gustav Tauschek fikk innvilget det første patentet på slik teknologi. Hans metode var en mekanisk anordning som benyttet templater som ved et eksakt motstykke sørget for at ikke noe lys ble sendt igjennom en lys detektor.

Teknologien har utviklet seg noe siden Tauschek, og i dag benytter i henhold til [41] alle OCR dataprogrammer nevrale nettverk i en eller annen grad. CNN har sitt utspring i OCR og MNist er et velkjent og velbrukt referanse datasett for ytelses måling av nevrale nettverksmodeller og er nettopp et datasett for trening av OCR kapabilitet. De nyeste og beste (convolutional) nevrale nettverkene oppnår en nøyaktighet med en så liten feilrate som helt ned mot 0,39% [124]. MNist består av 60000 håndskrevne tall og viser altså hvor flink en algoritme kan være til å kategorisere 10 forskjellige sammensatte former utifra et bilde på 28x28 piksler i gråtoner. For en oppdatert liste over resultater med nevrale nettverk som benytter MNist, se [120].

CENPARMI er et annet datasett som til forskjell fra MNist inneholder små (lowercase) bokstaver. [131] viser at feilrate resultater for forskjellige algoritmer som varierer fra 7,66 til 10,49 for MNist, varierer fra 10,7 til 16,63 for CENPARMI. Som ventet er feilraten altså større jo mer komplekse datasettene er. Et spørsmål her er om denne økningen i feilrate er en funksjon av kompleksiteten, eller om den er vilkårlig.

[137] beskriver et annet system som benytter CNN til OCR. De benytter Garbor filter til preprosessering og kaller modellen sin derfor for GCNN. Mot MNist datasettet oppnådde de en feilrate på 0,68%.

2.5.3 Generisk objekt gjenkjenning

[7], [15], [38], [47] og [123] er alle artikler som omhandler generisk objekt gjenkjenning. [47] introduserer en søkemotor for bilder på internett og viser at

MIR i 2000 var i en startfase. Viktig spørsmål i så måte er om dagens datamaskiner har kapabiliteter til å utføre MIR. Her er det to aspekter som må besvares for seg: Kan det søkes etter bilder med tekst per i dag, og/eller kan maskiner settes opp til å "forstå" innhold i bilder per i dag slik at de kan automatiseres til å utføre MIR på egenhånd?

Det første aspektet – kan det søkes etter bilder med tekst per i dag – kan delvis sees i lys av overgangen til Web 2.0 ([34]). Og her har det skjedd mye de siste årene, spesielt med hensyn til tagge kulturer. Internett steder som Flickr.com og YouTube.com har vokst frem og disse er basert nettopp tagger og kan dermed søkes igjennom ved hjelp av tekst. Vil gå nøyere inn på dette senere, men teknologi finnes altså så spørsmålet er om nok bruker slik teknologi til at MIR kan fungere effektivt.

2.5.4 Indeksering og søking av bilder på internett

Hvis nevrale nettverk er gode til å gjenkjenne objekter i bilder, burde det kunne antas at slike nettverk ville blitt benyttet på bilder på internett. Noen slike systemer finnes, men disse benytter gjerne ikke nevrale nettverk direkte.

Utifra patentsøknader fra Neven Vision ([19]) er systemet deres todelt. Første del av systemet benytter Gabor Wavelets til mønster ekstrahering mens den andre delen består av et nevralt nettverk og utgjør beslutningselementet for hva eventuelle mønsterfunn i et bilde representerer. Patent søknaden er dessverre for lite beskrivende til å vite nøyaktig hvordan systemet fungerer og det er heller ikke klart om Google.com, som nå er eiere av Neven Vision, har implementert dette systemet i deres image.google.com billedsøkemotor. Men hvis en skal spekulere vil det være naturlig å tenke seg at Neven Vision sitt system på ett eller annet vis er med i den underliggende strukturen til image.google.com, hvilket igjen fordrer at det kan kategorisere billedinnhold ved hjelp av tekstlig representasjon av objekter.

[137] beskriver forøvrig et annet system som benytter Garbor filter. De bruker det som tidligere nevnt som preprosessering til CNN for OCR.

Lew et al. ([47]) sin ImageScape søkemotor er et annet eksempel på en søkemotor for bilder. Men denne bruker ikke nevrale nettverk til å klassifisere objekter. Isteden benyttes Kullback relativ informasjons metode, med hvilket jeg går utifra de mener Kullback-Leibler avviks teori. Dette er en matematisk modell for å beregne avvik mellom informasjon.
Andre, som for eksempel [48], har derimot benyttet ImageScape som grunnlag for å benytte blant annet nevrale nettverks baserte løsninger til å gjenkjenne bilder. ImageScape, image.google.com og andre dedikerte søkemotorer kan sånn sett være et alternativ til å benytte sosiale nettverkssteder for opplæring av nevrale nettverk.

2.5.5 Spill

Et annet område som tidvis benytter både sosiale nettsteder og nevrale nettverk er spill. Her inngår gjerne nevrale nettverk i beslutningsprosessen i spill. [64] er et eksempel på metode for bruk av nevralt nettverk i spill. Det brukes her for å gjette seg frem til et objekt som spilleren tenker på. Det benyttes en original struktur på det nevrale nettverket – nemlig individuelle to dimensjonale (x og y) nettverks ”celler” satt sammen i en matrise. Nettverket fungerer som forovermatet, men kan benyttes begge veier. En måte å benytte det på er å bruke x-siden som inn og y-siden som ut. Den andre er altså å benytte y-siden som inn og x-siden som ut.

2.6 Annen relevant teknologi

2.6.1 Fusjonering av nevrale nettverk

[6] og [50] demonstrerer at det er mulig å kombinere mange nevrale nettverk til ett.

[6] beskriver et system med en hovedagent som sender ut replika av sitt nevrale nettverk til (mobile) mottaker agenter. Disse enhetene lærer sine respektive nevrale nettverk separat. Etter hvert returnerer de til den sentral løsningen hvor de kombineres til et nytt nevralt nettverk, typisk en ny hovedagent og prosessen kan repeteres. Sammensmeltningen av de nevrale nettverkene gjøres ved hjelp av de skjulte nevronene.

Dessverre ble ikke ytelsen forbedret med mindre det ble brukt samme datagrunnlag for alle de mobile agentene. Nytteverdien ligger derfor i parallellisering av nevralnettverkstrening og eventuelt for distribuert opplæring for gjenkjenning av forskjellige objekter. Altså; de mobile agentene kan settes til å løse ulike oppgaver basert på samme datasett.

[6] etterlyser i sin konklusjon flere metodikker for å kombinere nevrale nettverk.

[50] søker å løse et komplekst problem ved å dele det opp i del problemer og la forskjellige nevrale nettverk løse hvert av del problemene. Deretter kombineres de nevrale nettverkene til et nettverk som kan løse hele det komplekse problemet. Flere forskjellige metoder for å kombinere to nettverkene ble testet, og de resulterende nettverkene var som regel større enn summen av delene. [50] mener å ha vist at slik metodikk kan benyttes med positiv effekt ved parallell læring og for å kombinere enklere nevrale nettverk til komplekse varianter. Det største problemet med [50] er at de forskjellige nettverkene som skal kombineres må ha blitt trent opp på samme datasett. [50] sin løsning kan altså ikke benyttes slik beskrevet til en distribuert løsning for over heterogene miljøer eller forutsetninger. Dette blir altså mest en måte å parallell prosessere læring.

Verken [6] eller [50] ser altså ut til å kunne være gode løsninger for distribuert trening av nevrale nettverk på heterogene plattformer. Men [6] påpeker og viser at læring kan ekstraheres fra den skjulte strukturen som nevrale nettverk utgjør. Dermed kan det øynes håp om at fremtidig forskning vil kunne finne metoder som kan løse problemet med å kombinere heterogene nevrale nettverk.

2.6.2 Dynamisk utvidelse av nevrale nettverk

[28] argumenterer for og demonstrerer at ved ukontrollert læring (unsupervised learning) av et to lags nevralt nettverk kan effektiviteten økes ved å la læringsalgoritmen automatisk legge til nevroner med tilhørende vekter i utlaget. De finner dog at metoden kun fungerer for enkelte læringsteknikker, i deres tilfelle SOFM (Self-Organizing Feature Map – også kalt SOM eller Kohnen map), DSOFM (Dynamic SOM) og DFSCL (antar dette står for Dynamic Frequency Simple Competitive Learning). For SCL, DSCL og FSCL var virkningen heller motsatt. Men den negative effekten gjaldt kun for korrekt gjenkjenningsrate. For feilraten på sistnevnte metoder var det ingen store endringer noen vei.

Growing Neural Gas ([126]) er et annet eksempel på en dynamisk utvidbar nevral nettverks modell. Se [132] for et eksempel på virkemåte. Her opprettes (ut) nevroner for dynamisk å tilpasse seg predefinert inndata som fremstilles som en geometrisk form. Det interessante å se ved [132] er at den geometriske formen kan byttes underveis, og det nevrale nettverket vil tilpasse seg.

Har brukt [132] sin GNG-U til følgende lille eksperiment:

- *Ring*: 70’000 epochs til stabil tilstand med feilrate på 10±5.
  → *Large Spirals*: Etter 500’000 nye epochs har den fortsatt ikke tilpasset seg like godt som etter 90’000 epochs med *Large Spirals* som utgangspunkt. Ender opp med en feilrate på ca. 20±10.
- *Large Spirals*: 90’000 epochs til stabil tilstand med feilrate på 10±5.
  → *Large Spirals*: Etter ca. 50’000 nye epochs har den tilpasset seg godt til *Ring* med en feilrate på ca. 13±5. Men algoritmen klarer ikke å flyte eller fjerne alle nevroner som fra *Large Spirals* har havnet utenfor sirkelen. Etter totalt 250’000 epochs ligger det 59 nevroner inne i ringen mens det fortsatt er igjen 41 nevroner utenfor sirkelen (maks antall nevroner er satt til 100).

Altså kan det ligge en begrensning i algoritmen til [132] på muligheten fra å gå fra en tilstand til en mer kompleks tilstand. Men resultatet fra å gå fra en kompleks tilstand til en mindre kompleks tilstand tyder på at problemet vel så gjerne kan ligge i designet av algoritmen og ikke i selve nettverksmodellen.

Forøvrig er det verd å merke seg at GNG i utgangspunktet er laget for ukontrollert læring (unsupervised learning). [135] er et typisk eksempel på bruk av GNG. I denne artikkelen brukes metoden til å lage en modell av en (menneske)hånd. 16 bilder av hender i forskjellige positurer ble modellert i 2D.

Bildene var på 395x500 (thresholded – dvs. 2 bit fargeoppløsning), altså svært store i forhold til det andre nevrale nettverksmodeller klarer å undersøke etter dagens standard maskinvare kapabiliteter.

Konklusjon: Det later til at det finnes nevrale nettverksløsninger som dynamisk kan endre utparameter.

Anbefalt fremtidig forskning sett fra denne oppgavens problemstillings synsvinkel må være å kombinere Growing Neural Gas eller liknende med Convolutional Neural Network eller Pyramidal Neural Network. Spesielt interessant er PyraNet som benytter et regulært MLP som siste instans i sin modell. Hvis disse modellene kan kombineres skulle det altså tilsi at en vil kunne utføre mønstergjenkjenning med et dynamisk antall kategoriseringer.

2.7 Relevant tilgjengelig effektiviserings teknologi for nevrale nettverk

I 2.4.6 viste jeg at det relativt enkelt kan møtes en kapasitetsmessig barriere ved bruk av nevrale nettverk på store datamengder. Jeg vil nå se nærmere på noen teknologier som i vesentlig grad kan flytte slike barrierer.

2.7.1 Grid teknologi

I dag benyttes grid teknologi i mange forskjellige sammenhenger. Funksjonen til grid teknologi er å spre prosessering og av og til lagring. Dermed muliggjøres behandling av større datamengder enn det som ellers ville vært mulig. Blant slike oppgaver finner vi trening av nevrale nettverk med store mengder treningsdata. En måte å lage superdatamaskiner er å benytte grid teknologi for å distribuere arbeid mellom prosesserings enheter. Andre eksempler på bruk av grid teknologi er Folding@home og SETI@home.

Folding@home er et forsknings prosjekt tilhørende Stanford University og består i at folk laster ned et klient program som så får data oppgaver fra en server. Data oppgavene er i dette tilfellet beregninger for modellering av protein ”bretting”, det vil si hvordan proteiner genererer seg selv. Spesifikt blir dette for tiden brukt til kreft forskning. Per 23/9 2007 kjørte Folding@Home med en kapasitet på 1224 teraflops. Forøvrig står Sony Playstation 3 for 961 av disse teraflopsene men bare 15,2% av antall CPU’er. ([57], [59])

SETI@home er et forskningsprosjekt tilhørende Berkeley University. Dette prosjektet likner på Folding@home, men forskningen går på søk etter utenomjordisk intelligens. Dataene som lastes ned fra serveren av klientene er biter av innsamlede radio signaler. Per september 2001 oppe i 23,37 teraflops. ([58], [59])

Det Ian Foster et. al ([36], [39], [59], [60], [61]) ønsker å lage, er derimot ikke enkeltstående klient applikasjoner. De ønsker å lage *the Grid* som vil være et

slags nytt internett. Men istedenfor det internettet vi har i dag hvor det primære målet har vært utveksling av informasjon, ønsker Foster et. al å lage et nettverk for utveksling av prosesseringskraft og lagring. Altså en åpen grid som alle kan koble seg til.

[38] benytter seg av dette til trening av nevrale nettverk. De benytter seg av Globus Toolkit som er et forsøk på å realisere *the Grid*. [38] beskriver et prosjekt i startfasen, men beviser at ideen er realiserbar.

Forskning kan i fremtiden komme frem til effektive måter å kombinere flere nevrale nettverk. Foreløpig ser dette kun ut til å kunne være en alternativ måte å dele opp opplæringen på da det kreves samme treningsdata. Men i fremtiden kan det være mulig å kombinere nevrale nettverk med vidt forskjellig trening. Da kan grid teknologi være en mulig måte å ha ett felles nevralt nettverk som andre kan hente ut replika av for så å lære opp lokalt for siden å dele tilbake til andre brukere ved hjelp av Griden.

2.7.2 Maskinvare og parallell prosessering

For å øke ytelsen for prosessering av nevrale nettverk ved hjelp av maskinvare, benyttes maskinvare som kan gjøre flere operasjoner på en gang. Drastisk økning av maskinvare kapabilitet og parallell prosessering er derfor to sider av samme sak i denne sammenheng. La oss se på litt relevant forskning.

Superdatamaskin

La oss først ta et raskt blikk på hva som er mulig å gjennomføre med konvensjonell (seriell) prosessering på en maskin med en CPU i forhold til parallell prosessering på en såkalt superdatamaskin.

For 50x42 piksel inndata med to ut parametere for ansikts og kjønn gjenkjenning, trenger [1] 3259 trenbare parametere (og oppnår ved det en gjennomsnittlig klassifikasjonsrate på 95,6%). Dette kjørte på en Intel Pentium 4 på 2,8 GHz med 1 GB minne. For å konvergere mot ønsket klassifiseringsrate brukte denne maskinen 1,97 timer.

Til sammenlikning kjørte [43] $5*10^{10}$ trenbare vekter. For å gjøre dette hadde man til rådighet 4096 prosessorer med totalt ca. 2 TB minne (256 MB per prosessor og 1 TB fellesminne). Nå utførte ikke [43] sitt nettverk noen konkret oppgave, i hvert fall ikke som er beskrevet. Dette viser dog i det minste at maskinvare har noe å si for hvilke størrelser en kan ha på de nevrale nettverkene og dermed hvor komplekse oppgaver de kan settes til å løse.

GPU

[136] omhandler hvordan bruke CNN til OCR. Men den har en interessant vinkling, nemlig bruk av GPU (Graphical Processing Unit) – altså den aritmetiske prosessoren på skjermkortet – til å øke prosesseringshastigheten. GPU'er er på mange måter bedre egnet til nevral nettverks prosessering enn CPU'er – spesielt ved billedhåndtering. Det er to grunner til dette: For det

første har alle moderne skjermkort flere pipelines – altså mulighet til å prosessere flere regneoperasjoner på en gang. Fordel nummer to med GPU'er er at de er laget for 2D håndtering, mens CPU'er regner i en dimensjon. Altså kan GPU'er hvis brukt riktig mot billedbehandling jobbe dobbelt så rakt som prosessor hastighet og antall pipelines skulle tilsi. En tredje positiv ting som [136] ikke går inn på er at GPU'er gjerne har tilgang til minne som er raskere enn hovedminnet til CPU'en. Fortsatt er det vanlig med DDR2 minne i vanlige datamaskiner mens skjermkort – i hvert fall de med 48 pixel shader pipelines – kjører DDR3. Nå vil minnet til et skjermkort som regel være vesentlig mindre enn minnet til CPU'en, og det er mulig hastigheten på minnet har liten betydning i denne sammenhengen.

Processor name	Chip design	Type	GFLOPS
Athlon 64 X2 6000+	**AMD**	**CPU**	**18,5**
Core 2 Extreme QX6800	**Intel**	**CPU**	**37,7**
XBox 360 Xenon	**IBM**	**CPU**	**115,0**
Cell (Playstation 3)	IBM	CPU	218,0
XBox 360 Xenos	IBM	GPU	240,0
R580+	ATi	GPU	374,0
R600	**ATi**	**GPU**	**475,0**
GeForce 8800 GTX	**Nvidia**	**GPU**	**518,0**
X1900	**ATi**	**GPU**	**553,8**
RSX (Playstation 3)	Nvidia	GPU	1 800,0

Tabell 5: Kapasitet for forskjellige prosessorer hentet fra diverse kilder ([142])

Økningen i ytelse ved å bruke GPU'en istedenfor CPU'en til CNN håndtering, fant [136] til å ligge på 24% - 47%. Dette ble oppnådd med et NVIDIA GeForce 7800 Ultra skjermkort med 24 pipelines mens CPU'en var en Intel Pentium 4 (2.8 GHz). Det vil si et relativt nytt skjermkort mot en relativt gammel prosessor. En mer balansert sammenlikning hadde nok vært mot en Intel Core 2 Duo prosessor. ([136], [142])

Legg forøvrig merke til at CPU'ene med høyest ytelse selv benytter parallell prosessering. For eksempel består den raskete prosessoren tatt med i undersøkelsen over – Playstation 3 sin Cell prosessor – av syv kjerner.

Multikjerne prosessor

[138] er en annen artikkel som foreslår å løse prosesseringsbehovet til nevrale nettverk ved hjelp av maskinvare. De laget et dedikert prosessor kort med to prosessorer av typen NET32K, designet av Hans Peter Graf. Dette kortet klarte en over 100 milliarder (*billion*) multiplikasjon eller adderings operasjoner per sekund (legg samtidig merke til at denne artikkelen ble utgitt i 1995). Ikke bare det, men med CNN baserte LeNet oppnådde de 0,1% feilrate. Feil klassifiseringer ble forøvrig markert for menneskelig sjekk i etterkant. Det beste man kan oppnå ved hjelp av GPU'en til ATI Radeon X1950 er til sammenlikning 48 (pipelines) x 500 MHz = 24 milliarder operasjoner per sekund. Dog kan dette multipliseres med to til 48 Gflops ved egnede oppgaver fordi hver operasjon kan ta 2D. I tillegg er oppløsningen 32 bit oppløsning mot 1 bit for NET32K. Men allikevel er hastigheten de oppnådde med NET32K særdeles imponerende med tanke på at det har gått over 12 år og ren teoretisk

ytelse er fortsatt sammenliknbar. Nå kan selvfølgelig en GPU og en NET32K prosessor sammenliknes direkte, for de er høyt spesialiserte til forskjellige oppgaver, men dette sier ikke minst noe om hva som kan oppnås ved å produsere dedikerte prosessorer.

3 Sosiale nettsteder

”*...where other architectures have to fit within a context, an online community architecture* creates *that context*” [44]. Dette utsagnet fanger opp et viktig aspekt ved internettsamfunn, spesielt med hensyn til bruk til opplæring av nevrale nettverk. At innholdet til et internettsamfunn blir bestemt av brukerne betyr at eventuell opplæringen til et nevralt nettverk vil bli bestemt av brukerne. Det nevrale nettverket vil altså bli tilpasset brukerne og deres bruksmønster. Ved valg av sosialt nettsted må vi derfor gå nøye inn i intensjonene bak nettstedet for å få innsikt i hva slags læring vi kan forvente og dermed hvordan et resulterende nevralt nettverk vil fungere. Selv med tilsynelatende likt innhold kan intensjonene bak gi helt ulike resultater.

Foto.no er et nettsted med et sosialt aspekt ved mulighet for opplasting og diskusjon av bilder. Intensjonen bak Foto.no er utveksling og diskusjon rundt bilder spesielt med henblikk på fototeknikk, utstyr, fremgangsmåter, og lignende. Resultatet er at brukerne sine kategoriseringer vedrørende fototekniske aspekter er flere og mer fremtredende enn kategoriseringene vedrørende innholdsmessige aspekter. På Flickr.com er derimot intensjonen å dele bilder generelt. Her ligger hovedvekten av brukerne sine kategoriseringer på det innholdsmessige og i mindre grad på tekniske aspekter.

Vi kan altså gå ut ifra at et nevralt nettverk opplært til kategorisering på grunnlag av foto.no vil bli lært opp annerledes enn et opplært av Flickr.com fordi kategoriseringsgrunnlaget vil være annerledes. For å kunne forutsi hvordan de nevrale nettverkene vil utvikle seg vil det derfor være viktig å forstå underliggende mekanismer for hvordan sosiale nettsteder fungerer. Det vi er opptatt i denne sammenheng er funksjonalitet for kategorisering, og jeg vil i den sammenheng gjennomgå relevant forskning for tagging og for taksonomi.

3.1.1 Struktur over tid

[22] har undersøkt utviklingen av Flickr og Yahoo! 360 – to meget store sosiale nettverk med henholdsvis en og fem millioner noder (brukere) da undersøkelsen ble gjort. På grunnlag av dette og annen relevant forskning, søker de blant annet å si noe om utviklingen av sosiale nettsteder over tid generelt. Deres resultater viser at utviklingen med hensyn til brukere først går gjennom en kraftig vekst, etterfulgt av en negativ vekst og deretter en relativ lav men stabil positiv vekst. Det kan jo være nyttig informasjon å ta med seg hvis en skulle gjøre et forsøk på å lage et sosialt nettsted selv.

3.2 Hva er tagging?

[33] beskriver tagging på følgende måte: ”*Annotering av en ressurs som et bilde, en web side, osv. med et fritt valgt nøkkelord. Hvert nøkkelord kalles en tagg.*” (Fritt oversatt fra engelsk.)

Tagger er avhengige av den kognitive prosessen hos taggeren, det vil si hvordan taggeren i øyeblikket oppfatter det som tagges. Dermed blir viktige egenskaper i den kognitive prosessen også viktige for taggingen. Et aspekt vil være å sørge for at taggerens intensjon er noenlunde lik senere brukers intensjon. Et annet aspekt vil være å forsikre seg om at betydning – av taggen – oppfattes riktig.

Som eksempel; vi ønsker en kategorisering over mulig innhold i et bilde. Dette kan være elementer som landskap, portrett, båt, fugl, rev, og så videre. Da er det viktig at taggerens kognitive prosess fokuserer seg om innholdet i bildet og ikke andre aspekter som hvilken blenderåpning som ble benyttet. Dernest må taggenes betydning kunne oppfattes korrekt. For eksempel at en tagg som ”rev” blir tolket av systemet (oss) som riktig semantisk utgave av rev, det være seg et rovdyr, en langstrakt grunne, en røyk, eller annet.

3.2.1 Problemer med tagger

Lucy Suchman skriver i sin bok ”Plans and situated actions” ([5]) om problemer med kommunikasjon. På et metanivå er Lucy Suchman sin beskrivelse av typiske kommunikative problemstillinger svært treffende med hensyn til tagger. I nevnte bok skriver hun om robust kommunikasjon med referanse til Hayes og Reddy (1983) sitt arbeid. Robust kommunikasjon handler om å sørge for at den intensjonelle meningen blir overbrakt. I dette inngår blant annet å ta høyde for kontekstuelle forutsetninger, tvetydigheter, ufullstendig informasjon, og å forstå motiver til ”avsender”. Fra dette er det lett å se potensielle problemer ved tagging. [4] og [23] påpeker følgende:

- Polysemi – at ett ord kan ha flere betydninger.
- Synonymi – at flere ord har samme betydning.
- Hyponymi - at det finnes forskjellige abstraksjonsnivåer for et ord.

[4] demonstrerer hyponymi problemet ved å se på ordet kattedyr. Noen vil si at det grunnleggende abstraksjonsnivået her er katt, andre vil si dyr, atter andre vil si perser, og så videre. [4] skriver at løsningen til slike problemer gjerne ligger i å ha samme sosiale forståelse. Innenfor en gitt kultur eller sosial konstellasjon vil det ofte ikke benyttes mer enn en betydning for et gitt ord. Man vil gjerne ty til samme ord for å angi en gitt betydning og en vil benytte samme grunnivå for informasjonsutveksling.

3.2.2 Mulige løsninger på problemene med tagger

Det er vanskelig på generelt grunnlag å sørge for at alle har samme forutsetninger for sin tagging. En måte å redusere uønskede variasjoner på er å sørge for samme intensjon hos taggere. Dette kan gjøres ved at det sosiale nettstedet det tagges innunder har en veldefinert intensjon.

Videre kan strukturer blant tagger tre frem hvis nok individer bidrar. Ved mange nok tagger til en gitt ansamling av data vil flertallet av disse annoteringene være meningsfulle innenfor den rammen de er fremsatt i. Dette kalles gjerne for flokkens visdom (”wisdom of the crowd”) eller kollektiv visdom. Sammenholdt med intensjonen bak det sosiale nettstedet hvor en kollektiv tagg har blitt benyttet, har man en intensjonell og meningsfull tagg.

Dette er altså vesentlig i forhold til det å benytte tagger til å lære opp nevrale nettverk; for at en vilkårlig tagg med stor sannsynlighet skal være meningsfylt på et generelt grunnlag, må mange forskjellige brukere ha benyttet den. Videre må de ha benyttet den med samme intensjon som det én søker å lære opp det nevrale nettverket til å gjenkjenne.

Når man strukturerer store nok mengder av slik kollektiv tagging blir det gjerne kalt folksonomi ([4], [5], [33], [37]). Folksonomi er en forkortelse av ”folkelig taksonomi” og innebærer altså en strukturering av taggene.

Forskning viser at etter rundt 100 tagger på ett element stabiliseres som oftest bruksmønsteret av taggene. Dette underbygger at bruken av tagger over tid stabiliseres og dermed med hensikt kan struktureres. ([88], [33])

Det er altså viktig både å få samlet riktige kategoriseringselementer og å få frem riktig semantisk tolkning av disse. La oss først se nærmere på muligheten for kategorisering, og deretter om det kan løse det semantiske problemet.

3.2.3 Taggesky (tagcloud)

Taggeskyer er en vanlig form for visualisering av tagger, men kan også benyttes til maskinell organisering av dem. En taggesky er en todimensjonal representasjon av utvalgte ord fra en samling tagger. [83] skriver i sin artikkel av mai 2007 at det enda ikke er gjort eksperimentelle evalueringer av effektiviteten av taggeskyer. Men til tross for at vi enda ikke vet noe særlig sikkert om effektiviteten ved bruk av dem, er taggeskyer allerede i bruk i stor skala. Mange av de store nettstedene som Flickr.com, del.icio.us, youtube.com, last.fm, eller for den saks skyld dagbladet.no benytter taggeskyer i en eller annen form. De benytter slike taggeskyer på vidt forskjellige måter; fra å være en av hovedfunksjonene for utforsking av nettstedet til å kun gi brukere informasjon om hva andre brukere mener om en ressurs.

Det finnes mange måter å organisere en slik taggesky på. De viktigste elementene er:

- Alfabetisk sortering.

- Vilkårlig sortering.
- Semantisk sortering (ord med semantisk tilknytning settes i nærheten av hverandre).
- Benytte brukers preferanser for hvilke tagger som skal vises.
- Benytte popularitet som kriterium for hvilke tagger som skal vises.
- Endre skriftstørrelse etter hvor mye en tagg er brukt.
- Bruk av farger (eller gråtoner) for å skille mellom aspekter som semantikk og popularitet.

Popularitet kan representere enten antall ganger en tag har blitt brukt eller antall ganger et element har blitt tagget med en spesifikk tag. Selvfølgelig benyttes gjerne flere av disse organisatoriske metodikkene samtidig. ([23], [81], [85])

[81] konkluderer med følgende funn i sin undersøkelse av taggeskyer; alfabetisk sortering gjør at brukere søker fortere, fontstørrelse variasjon er et svært effektivt virkemiddel og taggenes plassering er også viktig. Videre fant de at brukere tilsynelatende ikke leser taggeskyer eller lister grundig, men heller bare ”kaster et raskt blikk” (”scans”). I så tilfelle vil brukeren fokusere på de uthevede ordene og først komme innom ord med mindre fontstørrelse ved grundigere lesning. [81] etterlyser mer forskning på nettopp dette. En anbefaling de gjør er å gjennomføre studier med øye sporing.

[84] på sin side mener at grupperings (”clustering”) teknologi i henhold til semantisk tilhørighet er den rette måten å lage oversiktlige og kompakte taggeskyer på. De argumenterer for at alfabetisk sortering kun er nyttig når brukeren vet hva han/hun leter etter.

[85] gir en god og grundig presentasjon av modeller og algoritmer for effektiv fremvisning av taggeskyer. De går spesielt etter 2D aspektet ved fremstilling av taggeskyer. For å kunne lage slike 2D fremstillinger påpeker de et problem; internettsider og dermed taggeskyer blir som regel skrevet i HTML, og HTML støtter i utgangspunktet fri posisjonering av tekst i et to dimensjonalt plan. Heller ikke CSS (Cascading Style Sheet) eller CSS2 gir god nok støtte til dette i henhold til [85]. De venter derfor på CSS3 som de håper skal overkomme disse hindringene.

Om man velger metodikk fra [81], [84] eller [85] er avhengig av, som de delvis inne på selv, bruk. Valg av design metodikk må stå i forhold til hensikten med det en lager. Taggeskyer kan være et verktøy som tas i bruk ikke bare ved søking men også for å gi ideer eller føringer til brukere av et sosialt nettsted for tagging. Ser vi dette med henblikk på problemstillingen for denne oppgaven kan en i visse tilfeller ha ønske om å styre brukere. Med Flickr som eksempel ønsker vi færrest mulig tagger i en folksonomi for å beskrive et spesifikt objekt.

3.3 Hva er taksonomi?

Taksonomi betyr klassifisering og brukes i denne sammenheng om strukturerte ordlister.

Taksonomier er vanlig brukt som klassifiseringsgrunnlag innen data og er i utgangspunktet rigide med de fordeler og ulemper det fører med seg. Taksonomier er hierarkisk av oppbygning. Utdrag fra taksonomier fremstilles gjerne i form av trestrukturer. Et vanlig eksempel å dra frem for å forklare hva taksonomi og hvordan det brukes, er indekserings systemer i biblioteker. Disse består av kategorier i en hierarkisk oppbygning etter en standardisert mal.

3.3.1 Eksempler på bruk av taksonomier

WordNet

[35] beskriver WordNet ([118]), hvilket er et eksempel på en taksonomi. WordNet er en leksikalsk database over det engelske språket som er tilgjengelig via internett. Videre er databasen bygget for å kunne benyttes ved hjelp av programmatisk kontroll. Nedlasting av hele databasen er fritt tilgjengelig for nedlastning fra Princeton universitetet.
Semantiske relasjoner som lagres i WordNet er:

- Synonymi (for subjektiver, verb, adjektiver og adverb)
- Antonymi (spesielt for adjektiver og adverb men også for subjektiver og verb)
- Hyponymi – at et ord er underordnet (subjektiver)
- Meronymi – at et ord er en del av (subjektiver)
- Troponomi – meronymi for verb (verb)
- Relasjoner (engelsk: entailment) (mellom verb)

To av de øverste klassene kjenner vi igjen som to av de tre store problemområdene for strukturering av tagger. Altså synonymi og hyponymi. Dette stemmer godt med [37] sitt utsagn om at det kan virke fordelaktig med blanding av taksonomi og tagge teknologi.

ODP

Open Directory Project ([119]) er et prosjekt basert på frivillige som har som mål å indeksere internett sider i en taksonomisk form. Det kan sies å være det taksonomiske motstykket til den tagge baserte del.icio.us. [24] viser at det er mulig å benytte ODP til høy presisjons søk etter innhold – ie. innhold på internett sider.

3.4 Hva bør velges – folksonomi eller taksonomi?

Som blant andre [4] påpeker er det store forskjeller på folksonomi og taksonomi.

Folksonomier bygger på ansamlinger av tagger er derfor dynamiske. Taksonomier er (til sammenlikning) statiske – fastsatt av utvikler eller eventuelt eier. Med andre ord; taksonomier skreddersys til eieres ønsker og behov mens folksonomier skreddersys til (flertallet) av brukeres ønsker og behov.

Når en skal velge en kilde til opplæring av nevrale nettverk er både taksonomier og folksonomier ønskelige. For på den ene siden ønsker en hele tiden å få størst mulig data grunnlag til opplæringen. Det betyr de mest populære taggene innen folksonomier. På den annen side ønsker en færrest mulig utnoder, for hver enkelt utnode betyr et betydelig større nettverk og betydelig større behov for opplæring. Altså ønsker man å forholde seg til en rigid taksonomi. Ved enkelte nevrale nettverk strukturer kan man faktisk bli bundet til en taksonomi, nemlig der hvor man ikke i etterkant kan forandre antall ut noder.

Dette faller godt sammen med [37] sin forutsigelse; løsningen ligger et eller annet sted imellom folksonomi og taksonomi. Ved å kombinere begge kan en i beste fall få i både pose og sekk; både brukerorientert, semantisk korrekt og balansert.

4 Data minering

Hva er dataminering? Hearst (1999) søker i sin artikkel å definere tekst data minering (TDM). I den sammenheng definerer hun dataminering til å praktiseres med to vidt forskjellige mål. Et mål er å predikere på grunnlag av data, et annet er å ekstrahere data. De som tilhører den første kategorien er de som ønsker å finne trender og mønstre ut ifra datamengder. Den andre kategorien, som er de som ”leter etter nåla i høystakken”, bedriver informasjonsekstrahering. Denne andre kategorien kaller hun ”nugget-finding” – altså utvinning av kostbar malm, eller informasjon i denne sammenheng, fra verdiløs masse ([25]). Et overordnet mål sies gjerne å være ekstrahering av ny kunnskap fra kjente data. ([71], [72])

Felles for dataminering uavhengig av mål er et etter hvert stort arsenal med verktøy. Dette inkluderer verktøy for å lage statistikk, klassifisere, gjenkjenne mønster og å predikere. Nevrale nettverksmodeller er et av disse verktøyene og kan brukes både innen klassifikasjon og prediksjon. Det finnes altså teknologi som strukturerer nevrale nettverk som et eget, atskilt verktøy, eller sett med modeller om en vil, på kjente og standardiserte plattformer.

4.1 PMML

En annen felles faktor innen dataminering er at det som regel utføres på eller mot DBMS’er (Database Management Systems). Ved bytte av DMBS er det gjerne ønskelig å få med seg datamineringsmodellene en har utviklet i tillegg til de rene dataene. For å kunne gjøre det har et konsortium kalt DMG (Data Mining Group) utviklet et språk i XML (eXtended Markup Language) form kalt PMML (Predictive Model Markup Language). Dette språket kan benyttes til å overføre modeller fra et DBMS til et annet DBMS, deriblant nevrale nettverks modeller.

DMG konsortiet består av kommersielle aktører som innbefatter blant andre IBM, Microsoft, Oracle, SPSS, SAS, SAP, NCR. Flere av disse har implementert støtte for PMML i produktene de produserer. [45]

Lager én et nevralt nettverk som kan beskrives og lagres i PMML, kan dette altså enkelt importeres i andre systemer. En kan dermed forestille seg å lage et system hvor et nevralt nettverk blir lært opp i et system for deretter å bli brukt i et annet system.

PMML krever et lag med nevroner for inndata som må være tall, og et lag med nevroner for utdata. Imellom disse lagene kan det være et vilkårlig antall lag med nevroner. Hvert enkelt nevron må ha en unik identifikator. Vekter legges

inn som del av et gitt nevron og består enkeltvis av et annet nevrons identifikator og en vekt. Det forutsettes at kun et lag regnes ut om gangen (en *epoch*). Det betyr at sykler tillates. Videre tillates også subsampling siden alle koblinger settes opp spesifikt. Convolutions er ikke eksplisitt støttet. Men siden kontakter kan spesifiseres til hvilket som helst lag betyr det imidlertid implisitt støtte for convolutions. ([79])

4.1.1 Begrensninger i PMML

En restriksjon med PMML er imidlertid hvilke aktiverings formler som kan spesifiseres. Følgende formler støttes:

- terskel: aktivering(Z) = hvis Z > grenseverdi så 1 ellers 0
- logisk: aktivering(Z) = 1 / (1 + exp(-Z))
- tanh: aktivering(Z) = (1-exp(-2Z)/(1+exp(-2Z))
- identitet: aktivering(Z) = Z
- eksponentiell: aktivering(Z) = exp(Z)
- omvendt: aktivering(Z) = 1/Z
- kvadrat: aktivering(Z) = Z*Z
- Gauss: aktivering(Z) = exp(-(Z*Z))
- sinus: aktivering(Z) = sin(Z)
- cosinus: aktivering(Z) = cos(Z)
- Elliott: aktivering(Z) = Z/(1+|Z|)
- arctan: aktivering(Z) = 2 * arctan(Z)/P_i

  hvor Z = Sum(W_i * utdata(i)) + bias

eller en såkalt radialBasis funksjon hvor det beregnes en euklidsk avstand vekter (på kontaktene) og inndata på denne formen:

aktivering(Z) = exp(f * log(vekt) - Z)

hvor Z = (Sum_i (utdata(i)-W_i)^2)/(2*bredde^2)

f = en summering av antall inndata til hver enkelt enhet i laget

vekt og bredde = er positivt tall som uavhengig av hverandre er unike for enten hver enkelt nevron, hver enkelt lag med nevroner eller hele det nevrale nettverket.

W_i = er en vektor lagret på kontaktene istedenfor en vekt.

Som følge av denne restriksjonen på aktiveringsformler kan muligens ikke alle nevrale nettverksmodeller beskrives i PMML. Det bør altså sjekkes hva som kan beskrives med PMML når en skal designe et nytt nevralt nettverk for å være sikker på at det i ettertid kan eksporteres.

En annen begrensning ved PMML er at den ikke støtter læringsmodeller. Standarden støtter imidlertid konsepter som taksonomi og vektor maskin. I tillegg er det basert på XML som jo er designet for strukturert overføring av data. Derfor er det en god basis for overføring av grunnlaget for en

læringsmodell. Men den programmatiske delen av en læringsmodell vil altså måtte bygges på nytt hvis en ønsker videre opplæring av det nevrale nettverket på det nye systemet. ([79])

4.2 Goebel and Grunewald's standard for klassifikasjon

Hvor utbredt er så PMML? I 2006 gikk jeg igjennom Goebel og Grunewald sitt survey fra 1999 og oppdaterte dette samt la til nye, viktige egenskaper ved database minerings verktøy. Det viktigste av disse tilleggene er PMML kompatibilitet. Andre tillegg jeg har gjort er:

- Operating system endret til Operating Platform
- Operating platform → Linux
- Operating platform → PalmOS
- Operating platform → Microsoft Windows CE
- Operating platform → SymbianOS
- Operating platform → Java
- DBMS (om verktøyet i seg selv er et database administrasjons system)
- Data sources → XML
- Data sources → IBM DB2
- Programmatically expandable
- Discovery Methodology → Naïve Bayes
- Discovery Methodology → Association
- Discovery Methodology → Time Series

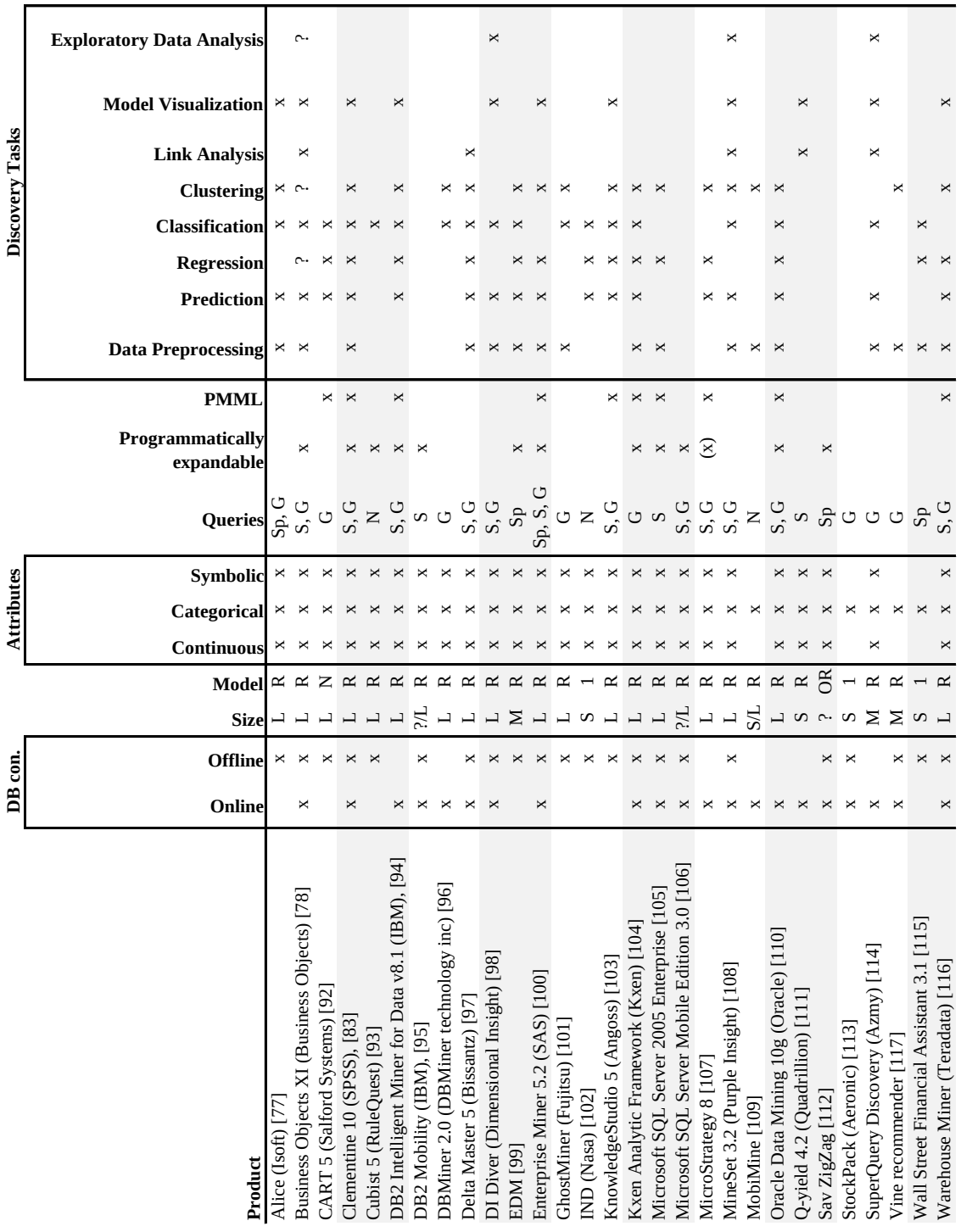
Følgende kolonner har jeg fjernet fordi jeg ikke lenger anser dem som relevante i henhold til det jeg har funnet av verktøy:

- Operating system → MF
- Operating system → Dos
- Operating system → Windows 3.x
- Data sources → DBase
- Data sources → Paradox
- Data sources → Foxpro
- Data sources → Lotus 123
- Discovery Methodology → Code Based Reasoning

Her følger en oversikt over funnene:

Product	Prod. Status	Legal Status	Acad. Licens	Demo	Architecture	Operating platform: Unix	Linux	Mac	MS Windows	MS Windows CE	PalmOS	Symbian	Java	DBMS	Data sources: Text	XML	Excel	Informix	Ingres	MySQL	Oracle	Sybase	MS Access	MS SQL Server	IBM DB2	ODBC
Alice (Isoft) [77]	P	C	N	R	S(, C/S)	(x)			x						x		x						x			x
Business Objects XI (Business Objects) [78]	P	C	N	N	C/S, P, D	x	x		x						x	x	x	x		x	x			x	x	x
CART 5 (Salford Systems) [92]	P	C	Y	Y	S	x	x		x						x		x							x		x
Clementine 10 (SPSS), [83]	P	C	Y	N	C/S, P, D	x	x		x						x	x	x	x			x	x		x	x	x
Cubist 5 (RuleQuest) [93]	P	C	N	Y	S	x			x						x	x										
DB2 Intelligent Miner for Data v8.1 (IBM), [94]	P	C	U	Y	C/S	x			x																x	
DB2 Mobility (IBM), [95]	P	C	U	Y	S, C/S		x		x	x	x	x		x						x					x	
DBMiner 2.0 (DBMiner technology inc) [96]	P	C	U	Y	S, C/S				x								x					x		x		x
Delta Master 5 (Bissantz) [97]	P	C	U	Y	S, C/S				x												x	x		x		
DI Diver (Dimensional Insight) [98]	P	C	N	Y	S, C/S	x	x		x						x					x	x			x		x
EDM [99]	P	C	N	N	S	x			x				x													
Enterprise Miner 5.2 (SAS) [100]	P	C	U	U	C/S	x	x		x																	
GhostMiner (Fujitsu) [101]	P	C	U	U	S				x						x		x	x			x		x	x		x
IND (Nasa) [102]	P	C	N	N	S	x	x								x											
KnowledgeStudio 5 (Angoss) [103]	P	C	N	R	S, C/S	x	x		x						x	x										x
Kxen Analytic Framework (Kxen) [104]	P	C	U	R	C/S										x	x	x							x		x
Microsoft SQL Server 2005 Enterprise [105]	P	C	Y	Y	S, C/S, P, D				x					x	x	x	x	x			x	x	x	x		x
Microsoft SQL Server Mobile Edition 3.0 [106]	P	C	Y	Y	S, C/S				(x)	x				x												
MicroStrategy 8 [107]	P	C	U	R	S, C/S	x	x		x												x	x		x	x	
MineSet 3.2 (Purple Insight) [108]	P	C	Y	R	C/S, P	x			x						x			x			x			x	x	
MobiMine [109]	R	U	U	U	C/S								x		x	x										
Oracle Data Mining 10g (Oracle) [110]	P	C	U	Y	S, C/S, P	x	x		x							x	x				x					
Q-yield 4.2 (Quadrillion) [111]	P	C	U	Y	S				x						x											x
Sav ZigZag [112]	P	F	N	N	S								x	x												
StockPack (Aeronic) [113]	P	C	N	Y	S						x				x											
SuperQuery Discovery (Azmy) [114]	P	C	N	Y	S				x						x		x						x			x
Vine recommender [117]	R	U	U	U	C/S						x		x		x											
Wall Street Financial Assistant 3.1 [115]	P	C	N	N	S					x	x				x											
Warehouse Miner (Teradata) [116]	P	C	U	U	C/S, P, D	x	x		x					x												

Tabell 6: Utvidet og fornyet Goebel and Grunewald's standard for klassifikasjon av dataminerings verktøy del 1/3.



	DB con.			Attributes				Discovery Tasks								
Product	**Online**	**Offline**	**Size**	**Model**	**Continuous**	**Categorical**	**Symbolic**	**Queries**	**Programmatically expandable**	**PMML**	**Data Preprocessing**	**Prediction**	**Regression**	**Classification**	**Clustering**	**Link Analysis**	**Model Visualization**	**Exploratory Data Analysis**
Alice (Isoft) [77]		x	L	R	x	x	x	Sp, G			x	x		x	x		x	
Business Objects XI (Business Objects) [78]	x	x	L	R	x	x	x	S, G	x		x	x	?	x	?	x	x	?
CART 5 (Salford Systems) [92]		x	L	N	x	x	x	G		x		x	x	x				
Clementine 10 (SPSS), [83]	x	x	L	R	x	x	x	S, G	x	x	x	x	x	x	x		x	
Cubist 5 (RuleQuest) [93]		x	L	R	x	x	x	N	x					x				
DB2 Intelligent Miner for Data v8.1 (IBM), [94]	x		L	R	x	x	x	S, G	x	x	x	x	x	x	x	x	x	
DB2 Mobility (IBM), [95]	x	x	?/L	R	x	x	x	S	x									
DBMiner 2.0 (DBMiner technology inc) [96]	x		L	R	x	x	x	G						x	x			
Delta Master 5 (Bissantz) [97]	x	x	L	R	x	x	x	S, G			x	x	x	x	x	x		
DI Diver (Dimensional Insight) [98]	x	x	L	R	x	x	x	S, G			x	x		x			x	x
EDM [99]		x	M	R	x	x	x	Sp	x		x	x	x	x	x			
Enterprise Miner 5.2 (SAS) [100]	x	x	L	R	x	x	x	Sp, S, G	x	x	x	x	x		x		x	
GhostMiner (Fujitsu) [101]		x	L	R	x	x	x	G			x			x	x			
IND (Nasa) [102]		x	S	1	x	x	x	N				x	x	x				
KnowledgeStudio 5 (Angoss) [103]		x	L	R	x	x	x	S, G		x		x	x	x	x		x	
Kxen Analytic Framework (Kxen) [104]	x	x	L	R	x	x	x	G	x	x	x	x	x	x	x			
Microsoft SQL Server 2005 Enterprise [105]	x	x	L	R	x	x	x	S	x	x	x		x		x			
Microsoft SQL Server Mobile Edition 3.0 [106]	x	x	?/L	R	x	x	x	S, G	x									
MicroStrategy 8 [107]	x		L	R	x	x	x	S, G	(x)	x		x	x		x			
MineSet 3.2 (Purple Insight) [108]	x	x	L	R	x	x	x	S, G			x	x		x	x	x	x	x
MobiMine [109]	x		S/L	R		x		N			x				x			
Oracle Data Mining 10g (Oracle) [110]	x		L	R	x	x	x	S, G	x	x	x	x	x	x	x			
Q-yield 4.2 (Quadrillion) [111]	x		S	R	x	x	x	S								x	x	
Sav ZigZag [112]	x	x	?	OR	x	x	x	Sp	x									
StockPack (Aeronic) [113]	x	x	S	1		x		G										
SuperQuery Discovery (Azmy) [114]	x		M	R	x	x	x	G			x	x		x		x	x	x
Vine recommender [117]	x	x	M	R		x		G			x				x			
Wall Street Financial Assistant 3.1 [115]		x	S	1		x		Sp			x		x	x				
Warehouse Miner (Teradata) [116]	x	x	L	R	x	x	x	S, G		x	x	x	x		x		x	

Tabell 7: Utvidet og fornyet Goebel and Grunewald's standard for klassifikasjon av dataminerings verktøy del 2/3.

Product	Discovery Methodology											Interaction		
	Neural Networks	Genereic Algorithms	Fuzzy Sets	Rough Sets	Statistical Methods	Decision Trees	Rule Induction	Bayesian Networks	Naïve Bayes	Association	Time Series	Autonomous	Human guided discovery	Highly Interactive
Alice (Isoft) [77]					x	x						x	x	
Business Objects XI (Business Objects) [78]	?	?	?	?	?	x	?	?	?		?	x	x	?
CART 5 (Salford Systems) [92]						x						x	x	
Clementine 10 (SPSS), [83]	x				x	x	x	x	x	x	x	x	x	x
Cubist 5 (RuleQuest) [93]						x	x					x		
DB2 Intelligent Miner for Data v8.1 (IBM), [94]	x					x				x	x	x	x	x
DB2 Mobility (IBM), [95]														
DBMiner 2.0 (DBMiner technology inc) [96]						x				x		x	x	
Delta Master 5 (Bissantz) [97]	x				x	x	x						x	
DI Diver (Dimensional Insight) [98]					x									x
EDM [99]	x					x				x			x	
Enterprise Miner 5.2 (SAS) [100]	x					x	x			x	x	x	x	x
GhostMiner (Fujitsu) [101]	x		x			x							x	
IND (Nasa) [102]						x						x		
KnowledgeStudio 5 (Angoss) [103]	x					x	x						x	
Kxen Analytic Framework (Kxen) [104]										x	x	x	x	
Microsoft SQL Server 2005 Enterprise [105]	x					x			x	x	x	x	x	
Microsoft SQL Server Mobile Edition 3.0 [106]														
MicroStrategy 8 [107]	x					x						x	x	
MineSet 3.2 (Purple Insight) [108]					x	x	x							x
MobiMine [109]	x				x	x		x				x	x	
Oracle Data Mining 10g (Oracle) [110]						x		x	x	x		x	x	x
Q-yield 4.2 (Quadrillion) [111]					x		x					x	x	
Sav ZigZag [112]														
StockPack (Aeronic) [113]					x								x	x
SuperQuery Discovery (Azmy) [114]					x		x					x		x
Vine recommender [117]	x											x		x
Wall Street Financial Assistant 3.1 [115]					x									x
Warehouse Miner (Teradata) [116]	x				x	x	x			x		x	x	x

Tabell 8: Utvidet og fornyet Goebel and Grunewald's standard for klassifikasjon av datamінerings verktøy del 3/3.

Det må understrekes at både [71] og min revisjon begge omhandler et relativt lite utvalg verktøy av hva som faktisk er tilgjengelig på markedet.
Noe som kjennetegner de store leverandørene som IBM, SAS, SPSS og Oracle er at de har mange spesialiserte verktøy. I disse undersøkelsene er derimot kun de generelle verktøyene tatt med.
Med hensyn til problemstillingen er det viktige i denne sammenheng at maskinlæringsalgoritmer i økende grad implementeres som en del av kjernefunksjonaliteten. Flere av de store DBMS leverandørene begynner å

levere grunnleggende støtte i kjerneproduktene sine. PMML støtte er en viktig indikator i så måte. Støttes PMML støttes grunnleggende data minering.
Alt i alt kan det konkluderes med at å lage maskinlæringsalgoritmer som kan beskrives med PMML gjør at algoritmene i etterkant vil kunne benyttes på en rekke plattformer. Det som da blir viktig er at inn- og utdata kan benyttes uavhengig av plattform. Altså vil det være viktig at modellen kan benyttes med så generiske data som mulig. For eksempel kan ikke "Gabor-wavelets" beskrives i PMML og dermed kan ikke systemer basert på [19] flyttes fra en plattform til en annen ved hjelp av PMML.

En mangel ved både [71] og min revisjon i denne sammenheng er at ikke MIR (Multimedia Information Retrieval) kapabilitet er undersøkt. En naturlig start for den teknologien jeg etterlyser, mener jeg vil være innen MIR. Selv har jeg tatt for meg billedkategorisering som et eksempel på bruk. [37] nevner nettopp klassifisering som et av problemene innen MIR. MIR blir stadig bedre integrert i informasjonssystemer og ikke minst også direkte i DBMS'er. MIR (*the search for knowledge in all its forms, everywhere*) passer godt innunder [25] sin definisjon av dataminering generelt. Derfor hadde det vært en verdifull utvidelse av [71] sin undersøkelse.
Videre hadde det lagt et godt grunnlag for å ta med interoperabilitet mellom plattformer. Med det mener jeg hvorvidt en kan flytte "verktøy", eller modeller om en vil, mellom modeller. Et eksempel på en slik kapabilitet for modell flytting er altså PMML. I henhold til [25] sin definisjon av data minering kan SQL språket være nok til å utvikle data minering verktøy (eller modeller om en vil), og dermed vil støtte for forskjellige SQL standarder være et annet eksempel på interoperabilitet.

5 Diskusjon

5.1 Grunnlaget

Jeg ønsker altså å lære opp nevrale nettverk. Det første elementet jeg vil ta for meg er grunnlaget som et slikt nevralt nettverk kan læres opp på. Her finnes flere muligheter. Man kan lage et slikt grunnlagsmateriell selv slik som [1] og [29] gjorde. Da får en akkurat det en ønsker og kan spesialtilpasse og optimalisere så mye en vil. Men det vil ta tid – spesielt hvis det er nevrale nettverk man er ute etter å lære opp siden disse krever mange unike eksempler for å bli effektive og robuste. [1] samlet inn hele 20'000 bilder. [29] samlet inn "bare" 1000 bilder, men måtte til gjengjeld bearbeide disse manuelt til de tilslutt hadde rundt 10'000 bilder.

En annen fremgangsmåte er å benytte eksisterende ansamlinger av data av ønsket type. Innen bilder finnes en rekke ferdige sett med spesifisert innhold. Bare innen databaser med bilder av ansikter kan nevnes FERET, UMIST, M2VTS og så videre ([31]). Fordelen med å benytte en "standardisert" database er at en da kan sammenlikne ytelse med andre løsninger som har benyttet samme database. På den annen side finnes det så mange forskjellige "standard databaser" at det kan ha begrenset nytte. Hvis en skal benytte slike databaser, bør en i så fall helst benytte flere. [31] har en sammenlikning mellom flere forskjellige modeller for ansiktsgjenkjenning i bilder, og siden mange dem har benyttet to standardiserte datasett gir det mening å sammenlikne med modeller som bare har brukt ett av datasettene.

En tredje fremgangsmåte er å benytte søkemotorer til å finne relevant informasjon for seg. [52] er en rapport med oversikt over slike søkemotorer med billedmateriale som spesialitet. Problemet med en slik fremgangsmåte er at mange, om ikke de fleste, av slike søkemotorer er internett baserte applikasjoner uten noe API (Application Programming Interface) som muliggjør enkel, programmatisk ekstraksjon av dataene. Altså vil en gjerne stå tilbake med manuell jobbing. I tillegg vil søkeresultatene gi svært varierende kvalitet (like variert som det som er tilgjengelig på internett naturligvis). Altså vil en igjen måtte gjøre en manuell jobb med å sortere ut bilder med riktig størrelse og format.

Mitt forslag i denne sammenheng er å benytte internettsamfunn. Potensialet for datagrunnlag vil da være et spørsmål om størrelse på samfunnet og aktivitet til medlemmene. Kvalitet vil selvfølgelig også være avhengig av brukerne. Altså vil en slik fremgangsmåte kunne slå begge veier, med mindre en kan finne et allerede eksisterende internett samfunn som passer til de kriterier en har til sin modell.

En viktig egenskap et slikt samfunn bør ha er et API. Det gir mulighet til å programmere inn automatikk i kommunikasjonen mellom samfunnet og det nevrale nettverket. Slik automatikk kan bestå i spesifisert uthenting av data. Som eksempel ved uthenting av bilder kan en tenke seg spesifisering av størrelse, antall, tidsperiode, rangeringer, og selvfølgelig kategoriseringstilhørighet, samt kategoriseringsinformasjon som nøkkeldata rundt bruksmønster og/eller semantisk betydning. Flickr og Facebook er begge internettsamfunn som har et slikt API tilgjengelig for brukere ([54], [55]). Selvfølgelig vil intensjonen bak et det respektive samfunnet være avgjørende for hvilke funksjoner som finnes for dettes API. Siden Flickr sin intensjon er sortering av bilder med tagger fritt valgt av brukere finnes det ikke noen funksjon for å direkte hente ut den semantiske betydningen av en tagg. Derimot finnes det en funksjon for å hente ut andre, relaterte tagger til en gitt tagg. Dette er ett eksempel på at automatisert opplæring av et nevralt nettverk selvfølgelig vil måtte ta hensyn til kildens muligheter og begrensninger.
Som konklusjon av dette mener jeg det må være riktig å dele inn et system basert rundt et nevralt nettverk i moduler. Kjernen med det nevrale nettverket vil i forskjellige faser ha forskjellige behov for inndata og produsere forskjellige utdata. I en oppstartsfase vil foring av data fra internett samfunnet være sentralt og utdata ikke ha noen verdi annet enn med hensyn til opplæringen. Senere kan en tenke seg både fortsettelse av opplæring fra internettsamfunnet, samt aktiv bruk med eller uten tilbakemelding til det nevrale nettverket både fra enkeltbrukere av internettsamfunnet, fra brukere med direkte tilgang, distribuert bruk, og så videre.

5.2 Avhengighet til ekspert baserte systemer

I de beskrivelsene av nevrale nettverksbaserte systemer jeg har sett på, har alle elementer vært beskrevet som et sammenhengende system. I noen systemer som [21] har dette vært en nødvendighet fordi det nevrale nettverket er spesialtilpasset preprosesseringen som igjen er spesialtilpasset datagrunnlaget. Derfor mener jeg LeCun et al. ([3], [7], [41]) gjør rett i argumentere for å lage nevrale nettverksmodeller som kan nyttegjøre seg data uten bruk av ekspert systemer i preprosesseringen. Å gjøre et nevralt nettverk avhengig av et ekspert system betyr to ting.

For det første vil en modell basert på preprosessering nødvendigvis medføre et spesialisert dataprogram for å ta seg av denne fasen. Det betyr at systemer plutselig er todelt, og modellen kan ikke flyttes fra en plattform til en annen uten også å ta med seg ekspert delen. Altså vil det ikke nødvendigvis være noe poeng i å gjøre det nevrale nettverket PMML kompatibelt med mindre preprosesseringen også kan gjøres tilsvarende kompatibelt. Et annet naturlig eksempel vil være et ønske om å implementere et ferdig system inn i et DBMS. Mange DBMS'er har per i dag støtte for PMML, men jeg har ikke funnet et

eneste DBMS som har *native* støtte for *Garbor wavelets*. Har en derimot et nevralt nettverkssystem som kun er en ansamling av data og struktur og ikke er avhengig av ekspert utviklet preprosessering kan en fritt flytte dette til DMBS'er som støtter benyttet struktur. CNN har ikke behov for *Garbor wavelets* da det er nettopp denne funksjonen *convolutional* lagene fyller. Videre vil bruk på andre plattformer ikke være en omprogrammering av et system som krever ekspert kunnskap, men en integrasjonsjobb.

For det andre er ekspert systemer i seg selv en begrensning da disse kun vil få ytelse så god som kunnskapen og ferdigheten til eksperten som lager det tillater. Et system basert på at det nevrale nettverket bygges direkte på rådata gjør isteden ytelsen til en funksjon av tid og kvalitet på inndata. Læringsteknikken vil fortsatt være avhengig av utvikling av ekspert kunnskap, men risikoen her går i henhold til forskningen i retning av prosesseringskrav og tidsbruk for konvergens mot maksimal ytelse. Når et ekspertsystem tar seg av mønsterekstrahering som i [21], vil det medføre feilmarginer med hensyn til systemets feilrate allerede i preprosesseringsstadiet. I tillegg må fortsatt et system med ekspertbasert preprosessering inkludere en læringsteknikk som igjen vil være avhengig av ekspert kunnskap. ([41])

Dette kan summeres i et lite diagram:

	Fordeler	**Ulemper**
Nevralt nettverk basert på prosessert data ved **ekspert kunnskap**	- Raskere optimalisert ytelse. - Mindre behov for trening. - Mindre krevende med hensyn til maskinkraft.*	- Ekspert kunnskapen vil avgjøre ytelse**. - Plattform avhengig.
Nevrale nettverk basert på "**rådata**"	- Plattform uavhengig grunnet - All kunnskap lagret i det nevrale nettverket	- Krevende å trene.*** - Kan bli svært maskinkraft krevende.
* Ikke minst fordi det nevrale nettverket vil være mindre da mønster gjenkjenningen ikke vil ligge det nevrale nettverket. ** Med ytelse menes både feilrate for systemet – altså ytelse med hensyn til å løse problem – og ytelse med hensyn til hastighet. *** Med hensyn til mengde data som trengs.		

Tabell 9: Fordeler og ulemper ved bruk av generiske nevrale nettverk kontra ekspert baserte nevrale nettverk

5.3 Tagging

Jeg har gjort en subjektiv undersøkelse av bruk av tagger i Flickr med hensyn til hvordan denne bruken kan gi representative kategoriseringsgrunnlag for nevrale nettverk. Det subjektive aspektet ligger i kategoriseringen av et bilde som enten korrekt, delvis korrekt eller ukorrekt/feil. Ved å kategorisere et bilde som korrekt mener jeg altså at et bilde inneholder en fullt gjenkjennbar representasjon av det jeg mener en gitt tag representerer. Jeg satte løst opp følgende kriterier for de forskjellige taggene:

- *Face*: Korrekt representasjon må inneholde enten to øyne, nese og munn eller hele hodet og minst to av munn, nese eller øye. Kun ansikter i noenlunde fokus av mennesker ansees som korrekt.
  Delvis korrekt innbefatter også dyre ansikter og representasjoner av ansikter som tegninger, figurer og liknende. Ansiktene trenger ikke være fokuserte eller tydelige.
  Ved ukurant eller feil har jeg ikke kunnet se noe som tydelig ligner på et ansikt.
- *Bicycle*: Korrekt vil si at en sykkel er godt synlig i bildet og tatt tilnærmet vinkelrett på sykkelen slik at man ser begge hjulene. Ved ukurant kan ingen hjul klart skjelnes, eventuelt ingen sykkel i det hele tatt. Delvis er de bildene som ikke faller i de andre to kategoriene.
- *Beach*: Vann og land må sees og ta opp mesteparten av bildet for å være korrekt. Delvis tillater andre objekter i bildet så lenge det ikke dominerer helt. Ved ukurant kan en i beste fall skimte vannskillet.
- *Flower*: Stort sett enkel å skille på. Et problem med korrekt klassifisering var om mer enn en blomst skulle tillates hvilket jeg gjorde. Dette kan godt sies å være en feil avgjørelse da det finnes en egen tagg for flertall – *flowers*. Nærbilder hvor bare deler av blomsten fylte hele bildet havnet i delvis. Ukurante inneholdt ingen synlig blomst.
- *Sky*: Korrekt – bare himmel. Ukurant – maksimalt noen blå flekker. Delvis – det imellom.
- *Coin*: Tillot bilder av skadede mynter som korrekt, samt flere mynter på et og samme bilde. Ukurante inneholdt ingen mynt eller kun deler av mynt som da gjerne representerte noe annet som for eksempel en fisk.
- *Landscape*: Valgte natur bilder med dybde som basisnivå. Her kunne man satt basisnivå til å gjelde for eksempel kontorlandskaper i stedet, men taggerne var tilsynelatende stort sett enig med natur som utgangspunkt. Ukurante vil gjerne si by eller portrett bilder. Delvis bilder inneholder flere elementer enn kun natur.

Tag	Antall tagger	Korrekt		Delvis korrekt		Ukurant eller feil	
Face	296 326	223	64,8 %	91	26,5 %	30	8,7 %
Bicycle	272 583	66	28,0 %	85	36,0 %	85	36,0 %
Beach	2 703 604	78	55,7 %	26	18,6 %	36	25,7 %
Flower	1 668 744	162	81,0 %	19	9,5 %	19	9,5 %
Sky	1 526 412	86	78,2 %	12	10,9 %	12	10,9 %
Coin	19 130	107	48,6 %	29	13,2 %	84	38,2 %
Landscape	917 871	160	75,5 %	20	9,4 %	32	15,1 %
Landscape + Nature	166 155	98	89,1 %	11	10,0 %	1	0,9 %
Bike + Bicycle	175 272	31	26,1 %	58	48,7 %	30	25,2 %

Tabell 10: Oversikt over funn gjort med vedlegg 8.3 og vilkårlig valgte tagger.

Tabell 10 viser en oversikt over de funnene jeg har gjort. Et av de gjennomgående problemene ved bilder og tilhørende tagger i Flickr, er opplasting av billedserier hvor mange innholdsmessig ulike bilder tagges likt. Typisk er enkeltbrukere som laster opp alle bilder fra en begivenhet. For eksempel alle bilder fra en sykkeltur hvor alle bilder blir tagget med *bike*, *bicycle*, feriested, hvem man dro sammen med, og så videre. Et annet eksempel er tur hvor tagger tydelig synes det er mye fint landskap rundt omkring og alle bilder dermed får taggene *landscape* og *nature* selv om et vesentlig antall bilder er av reisefølge, butikkvinduer og annet som ikke kan sies å ha noe med *landscape* eller *nature* å gjøre.

Dette fenomenet kan sees på som en form for problemet hyponymi, at forskjellige folk har forskjellige forutsetninger for sin kommunikasjon og dermed har et annet basis nivå for de ordene de benytter. Altså; noen registrerer *bicycle* som et grunnivå for en turform, mens andre bruker det for sykler og atter andre bruker det for sykkeldeler, og så videre.

En foreslått metode av blant andre [5] for å bøte på hyponymi er å utveksle flere ord om samme tema for at mottaker skal kunne forstå hvilken heuristisk ramme som skal være gjeldene for utsagnet. For å undersøke dette antok jeg at hvis det ble benyttet synonymer til den betydningen av den originale taggen som det ble søkt etter, ville dette bekrefte at taggeren faktisk har samme definisjon som det som søkes etter.

En annen vinkling som er en annen fremgangsmåte for å oppnå samme resultat er å benytte en taksonomisk ordliste til å finne flere ord som beskriver det samme. Til dette forsøket benyttet jeg WordNet beskrevet tidligere under avsnittet om taksonomier. Der fant jeg at *plane* også kan beskrives med *aircraft*, at *cat* kan beskrives med *feline*, og at *nose* kan beskrives med *olfactory organ*.

En tredje vinkling var at hvis en tagger benytter flere tagger, så er det en fare for at en ressurs inneholder mer en det en spesifikt er ute etter.

Vedlegg 8.3 viser et program skrevet i C# for lett å kunne utføre en slik undersøkelse mot Flickr. En viktig oppdagelse var at søk utført via API'et til Flickr ikke gav samme resultater som å gjøre søkene via web portalen.

Resultatene vises i Tabell 11. Ikke veldig overraskende var det faktisk ingen brukere som hadde benyttet en kombinasjon av *nose* og *olfactory*. Allerede her ser vi altså at en må ta seg i akt ved sammenblanding av taggeteknologi og taksonomi.

Å kombinere tagging og taksonomi ser allikevel ut som en positiv fremgangsmåte for mulig filtrering av data. Skal en slik sammenkobling være effektiv ser det dog ut til at den må ha et aktivt element, altså til en viss grad styrt av brukere. Her vil det meste av jobben ligge i å få et effektivt. Vedlegg 8.3 er ikke knyttet til noen form for sjekk mot taksonomi eller tagging utover eksakt eller implisitt treff på forekomst av taggen(e) det søkes etter. All kobling mot WordNet og Flickr.com sine taggeskyer er her gjort manuelt. I et anvendelig system må valgene settes opp mot mengden returnerte bilder. Når det ikke returneres noen bilder for kombinasjonen *nose* og *olfactory*, betyr ikke det at det ikke finnes bilder i Flickr som er kun av en nese. Dermed må en bruker kunne innsnevre eller utvide krav til koblinger avhengig av antallet og kvaliteten på bildene applikasjonen henter ut.

Tags	Exact match required	WordNet	% correct	Correct	False	Out of false; contains fragments of correct	Tags to correct pictures					Tags to false pictures					Number of pictures for tag	Pictures searched
							1	2	3	4	5+	1	2	3	4	5+		
cat	No	No	78 %	39	11	11	17	4	3	4	11	3	2	1	1	4	1 286 866	50
cat	Yes	No	96 %	48	2	2	48	0	0	0	0	2	0	0	0	0	1 286 866	100
cat,black	No	No	68 %	34	16	10	0	2	6	1	25	0	0	1	1	14	43 875	50
cat,blue	No	No	-	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-
cat,feline	No	Yes	72 %	36	14	14	0	0	0	0	36	0	0	0	0	14	55 272	50
cat,feline	Yes	Yes	-	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-
cat,striped	No	No	70 %	35	15	15	0	0	0	1	34	0	0	0	0	15	832	50
cat,striped	Yes	No	100 %	1	0	0	1	0	0	0	0	0	0	0	0	0	832	832
cat,white	No	No	-	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-
eye	No	No	22 %	11	39	21	0	0	1	1	9	1	2	4	2	30	145 401	50
eye	Yes	No	92 %	46	4	1	50	0	0	0	0	0	0	0	0	0	145 733	1 723
face	No	No	70 %	35	15	14	1	0	1	0	33	0	0	0	0	15	281 344	50
Face	Yes	No	42 %	21	29	8	50	0	0	0	0	0	0	0	0	0	281 321	4 400
nose	No	No	4 %	2	48	35	0	0	0	0	2	0	0	2	1	45	46 908	50
nose	Yes	No	6 %	3	47	47	50	0	0	0	0	0	0	0	0	0	46 900	2 805
nose, olfactory organ	No	Yes	-	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-
plane	No	No	80 %	40	10	2	0	0	0	1	29	0	1	0	2	7	174 061	50
plane	Yes	No	42 %	21	29	4	21	0	0	0	0	29	0	0	0	0	174 061	3 935
plane,aircraft	no	Yes	28 %	14	36	7	0	0	0	0	14	0	0	0	0	36	29 181	50
plane,aircraft	Yes	Yes	100 %	1	0	0	0	1	0	0	0	0	0	0	0	0	29 181	29 181
Sofa	No	No	40 %	20	30	18	2	12	1	0	6	0	0	0	5	23	24 180	50
Sofa	Yes	No	74 %	37	13	10	50	0	0	0	0	0	0	0	0	0	24 180	1 508
sofa,red	No	No	2 %	1	49	13	0	0	0	1	0	0	0	0	1	48	1 048	50
sofa,red	Yes	No	0 %	0	1	1	0	0	0	0	0	0	0	0	0	1	1 048	1 048
tree	No	No	58 %	29	21	13	0	1	1	0	27	0	0	0	0	21	876 107	50
tree	Yes	No	88 %	44	6	5	44	0	0	0	0	6	0	0	0	0	876 107	3 653

Tabell 11: Søk fra Flickr utført med 8.3 og testet med og uten manuell bruk av WordNet

5.4 Nødvendigheten av nøyaktighet for et nevralt nettverk

Det er viktig å merke seg at krav til nøyaktigheten til et nevralt nettverk vil variere kraftig. Innen OCR regnes 99+% som tilfredsstillende mens for uthenting av bilder fra internett vil en med søkeredskap som ImageScape og Flickr oppnå et sted mellom 50%-90%. Men det betyr jo ikke at OCR programvare benyttes mer enn Flickr. Ergo; krav varierer.

Krav vil også variere med tid, og da spesielt for typiske maskinlærings-algoritme oppgaver. For eksempel OCR vil i en startfase tillates en viss feilrate fordi en må regne med en viss tilvenning av algoritmen til spesifikk fonttype, layout, lysstyrke, og så videre. Men etter hvert vil kravet til korrekt kategorisering øke. Hvis en OCR algoritme har en gjenkjennelses rate på 99% vil det for denne oppgaven sin del bety at en ved skanning av denne teksten vil måtte svare på spørsmål eller få feil kategorisering på i snitt 15,5 bokstaver per side. Det siste CNN baserte systemet har en gjenkjennelsesrate på 99,61%. ([120], [124])

For talegjenkjenning antydes det av [15] at en gjennomsnittlig gjenkjenningsrate på 98,5% er akseptabelt, mens en rate på 93,7 er heller dårlig.

For ansikts*deteksjon* oppgir [31] (fra 2002) ”state of the art” metodikker til å ligge på 74,2% til 98% med de beste på rundt 94%. Systemene med 94% er bedre enn det med 98% fordi sistnevnte får en veldig høy grad av falsk deteksjon.

Som nevnt tidligere gir det liten mening å sammenlikne noen som helst av disse resultatene. Først og fremst fordi datasettene vil være ulike fra en type gjenkjenning til en annen, men også fordi de forskjellige systemene er svært forskjellige med hensyn til modeller og teknikker som benyttes. For eksempel har et av de aller beste resultatene for gjenkjenning av generiske objekter fra direkte input av virkelighetsnære bilder en feilrate på 5,9%. Men læringsmetoden som da ble benyttet var SVM som ofte har vært praktisk umulig å benytte grunnet de store kravene til maskinvare kapasitet. [123]

5.5 Valg av læringsteknikk for nevrale nettverk

Hvilken type læring en velger å benytte er avhengig av de praktiske forutsetningene og konsekvensene av implementeringen. Nok et positivt aspekt ved å benytte rådata som input er sågar at man ikke nødvendigvis er avhengig av én læringsteknikk.

Generelt gjelder som tidligere nevnt én regel for læring; nemlig at jo bedre og raskere, desto mer krevende maskinvaremessig. For de aller beste teknikkene

kommer dette av at de bygger på kalkulasjoner over hele læringsdatasettet samtidig. Med NORB treningssettet som totalt teller 617'220 bilder av 96x96 piksler (i gråtoner) betyr det altså 5,3GB minne. Men som [123] skriver flyttes stadig grensene for tilgjengelig maskinvare, og 5,3GB er mulig å oppnå med "normale" datamaskiner allerede. Men når disse 5,3GB må multipliseres med antall vekter og nevroner (størrelses orden fire millioner i denne sammenheng) samt adderes til antall vekter og nevroner i annen potens, er det fortsatt noen år til den svært effektive læringsmodellen Levenberg-Marquardt kan brukes til for eksempel generisk objektgjenkjenning av råbilder fra digitale kamera som. Og da snakker vi uansett digital kamera – selv VGA oppløsning.

Men dette gjelder egentlig bare for initiell trening for et slikt system som denne oppgaven tar sikte på. I bruk – fortrinnsvis med tilhørende tilbakemelding til systemet – vil et treningssett ikke inneholde mer enn normalt en til maks 100 enheter (se vedlegg to). Det gir altså helt andre forutsetninger til hvilke læringsmetoder som kan benyttes. På den ene side står en friere til hvilke teknikker en ønsker å benytte, på den annen side vil ikke avanserte læringsmodeller som Levenberg-Marquardt kunne benyttes. De mest avanserte læringsmetodene er jo avhengig av å ha *hele* treningssettet tilgjengelig for å være effektiv.

Et annet aspekt som jeg ikke har funnet noe svar på og som burde være et interessant forskningstema er hvordan treningssett kan deles opp ved trening av nevrale nettverk med mer enn en utnode. Kan et datasett deles opp for hver kategori som ønskes opplært, eller vil en ny slik trening overkjøre og ødelegge eksisterende trening? Et slikt eksperiment vil ikke nødvendigvis være representativt for mer enn den nevrale nettverksmodellen som blir testet. Poenget med CNN er å lagre individuelle, faktoriserte mønstre i de forskjellige *convolutional* lagene. Så selv om trening av forskjellige kategorier med separate datasett kan ha en negativ innvirkning på MLP kan muligens et CNN fortsatt dra nytte av det i sine mønster ekstraherings lag.

5.6 Praktiske eksempler – hvor vanskelig kan det være?

Vedleggene til denne oppgaven er kildekode for eksempler til delene et virkelig system som denne oppgaven beskriver vil trenge. Vedlegg en viser hvordan en teoretisk kan bygge opp et enkelt MLP og hvor vanskelig dette kan være. Vedlegg to er en enkel, fungerende MLP applikasjon. Vedlegg tre er et enkelt program for å hente ut bilder fra Flickr.com. I vedlegg 4 har jeg på en svært enkel måte kombinert det enkle MLP'et med uthenting av bilder fra Flickr.

Til et fungerende system slik beskrevet i denne oppgaven, vil en videre trenge en eksport funksjon til PMML. Vedlegg en inneholder alle elementene som behøves for en PMML eksport og ingen elementer som ikke støttes av PMML.

5.6.1 Et tilsynelatende korrekt eksempel

Vedlegg 8.1 er skrevet i programmeringsspråket Lua ([62]). Grunnen til at jeg valgte Lua er for det første at det er raskt både med hensyn til kjøring, programmering og kompilering, samt at det finnes (native) kompilatorer for mange plattformer. En løsning i Lua er for eksempel enkel å få testet på en (relativt sett) treg, mobil plattform som for eksempel PalmOS.

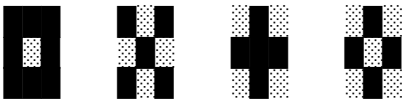
Koden er satt til å løse det klassiske XOR problemet. Fire former for inndata er gitt; 0 og 1 eller 1 og 0 eller 0 og 0 eller 1 og 1. De respektive svarene er 1, 1, 0 og 0. Koden er basert på nettverksmodellene og bakover propageringsalgoritmene gjengitt i [133] og [40]. Sistnevnte angir til og med alle vekter og verdier for første iterasjon samt resultatet for første epoch. I tillegg angis det hvor mange epochs som trengs for å divergere mot ønsket svar avhengig av relevante verdier satt for bakover propagerings algoritmen.

Til tross for at koden i vedlegg en tilsynelatende replikerer verdiene angitt i [133], divergerer alle utdataene fra nettverket mot 0,5 etter bare ca. 100 epochs uavhengig av hvordan initielle vekter er satt. Fremgangsmåten og det matematiske grunnlaget beskrevet i [133] er forøvrig svært likt bakover propageringsalgoritmen beskrevet i [40].

Dette eksempelet viser hvor ømfintlige nevrale nettverk kan være.

5.6.2 Et fungerende eksempel

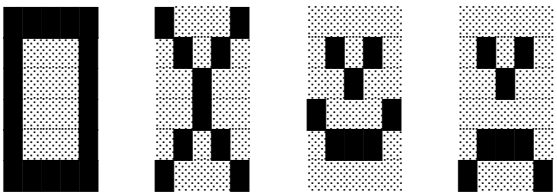
Vedlegg 8.2 viser en enkel applikasjon hvor det benyttes et MLP med ett skjult lag nevroner. Inndata bestod i første test datasett av ni tall som er null eller en og satt i fire forskjellige, enkle mønstre som følger (■ = 1, ▒ = 0):



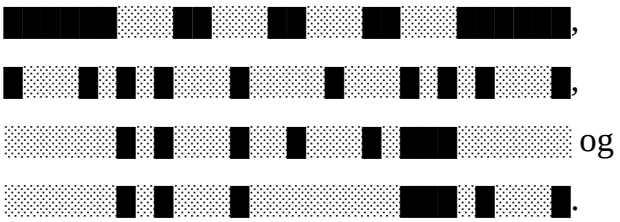
Legg forøvrig merke til at det nevrale nettverket ikke har noen form for dimensjoner innbygget, så det som egentlig mates inn i det nevrale nettverket er:

Det nevrale nettverket ble altså satt til 9 nevroner i innlaget og fire nevroner i utlaget. Det skjulte laget ble satt til 36 nevroner. Totalt antall vekter ble 468. På en AMD Athlon 64 3200+ (2,01 GHz) prosessor tok hver epoch i snitt ca. 1,4 millisekunder. Algoritmen er nøye bygget opp etter Rumelheart, Hinton og William sin modell.

Det andre testsettet besto av fire mønstre bygget opp av 30 tall som følger:



Eller som datamaskinen ser det:


og

For dette andre datasettet ble antall vekter 1224 det brukt ca. 3,34 millisekunder på hver epoch.

I Tabell 12 vises en oversikt over en del kjøringer av programmet. Viktig her at det ikke er testet mot annet enn det datasettet som det er trent opp i. Hver utnode trenes altså opp med et positivt og tre negative eksempler for hver trenings iterasjon. Det betyr altså at det nevrale nettverket spesifikt er trent opp til å returnere 0 på tre av inndataene så vel som å returnere 1 for de resterende inndataene.

Videre har jeg aldri stilt på standardverdiene for epsilon eller bias som man kan se i koden er satt til 0,1. Ved å stille disse verdiene kan både suksessrate, klassifisering og konvergeringstid ”fin-tunes”. Mine tester viser altså resultater fra ikke-optimaliserte kjøringer. Går vi utifra resultatene til [133] som jo ikke har en helt ulik nevral nettverksstruktur som det jeg benytter i, kan justering av epsilon alene resultere i opptil 50 ganger så rask konvergering. Men siden økning av disse verdiene vil medføre minsket suksess rate, har jeg valgt å la de stå på ”standard” verdier. Det er forøvrig vanlig å la brukere få tilgang til å justere disse tallene i slike applikasjoner, og det ser jeg som en naturlig utvidelse av mine programmer skal disse videreutvikles.

Fra tabellen ser vi at i samsvar med det [40] sier, er det ikke alltid at nevrale nettverk konvergerer mot et ønsket resultat. I dette tilfellet er det også jo høyere mål for nøyaktighet som settes, desto større er sjansen for at den ikke vil konvergere i det hele tatt. Selv om man ser suksess raten i forhold til antall epochs som faktisk har blitt foretatt er fordelingen ujevn.

Number of input nodes	Target MSE	Success	Average values: Succeeded: Epochs	Succeeded: MSE	Succeeded: Classification Best	Succeeded: Classification Worst	Failed: Epochs	Failed: MSE	Failed: Classification Best
9	0,0100	99,50 %	1 232	0,0098	93,5 %	17,4 %	30 000	0,1377	96,6 %
9	0,0050	97,60 %	1 967	0,0049	95,7 %	12,8 %	30 000	0,1365	92,6 %
9	0,0010	86,80 %	5 741	0,0010	98,1 %	6,4 %	30 000	0,1214	84,9 %
9	0,0010	88,00 %	6 186	0,0010	98,0 %	6,6 %	30 000	0,1111	88,3 %
9	0,0005	72,70 %	8 834	0,0005	98,6 %	4,2 %	30 000	0,1085	89,1 %
30	0,0100	100,00 %	829	0,0098	92,3 %	16,3 %	N/A	N/A	N/A
30	0,0050	99,40 %	1 372	0,0048	95,8 %	12,4 %	30 000	0,1813	62,0 %
30	0,0050	99,70 %	1 472	0,0048	95,7 %	12,6 %	30 000	0,0921	98,5 %
30	0,0010	92,00 %	4 227	0,0010	98,3 %	6,4 %	30 000	0,1064	88,5 %
30	0,0005	82,00 %	6 982	0,0005	98,8 %	4,7 %	30 000	0,1034	91,2 %

Each line represents 1000 individual training sessions

Tabell 12: Gjengivelse av snitt av kategoriseringsrater gjort med kjøringer av 8.2.

Jeg fikk engang spørsmål om hvorfor ikke nevrale nettverk benyttes over alt hvis de er så fortreffelige. Da jeg stilte spørsmålet videre til professor Herman Ruge Jervell var hans svar at ekspert systemer/matematiske modeller gjerne konvergerer fortere og er mer stabile enn nevrale nettverk. Første amanuensis (associate professor) Maia Dimitrova sitt svar var sitat; *"Neural networks (...) can learn anything to any degree of precision (!!!!! the most important theoretically proven property)."* [40; side 603] skriver at nevrale nettverk kan - men ikke nødvendigvis alltid vil - skape et hyperplan som kan klassifisere både lineære og ikke-lineære adskillbare data. Usikkerheten er prisen som betales for muligheten til å løse de ikke-lineære klassifiseringene

Nå har ikke jeg testet noen matematisk basert modell her, men de testresultatene som her er gitt avkrefter ikke professor Jervell sin påstand. Derimot demonstrerer mine resultater det [40] sier, og til en viss grad Dimitrova sin uttalelse. Ved en (og så vidt jeg har kunnet se – kun en) anledning har algoritmen faktisk returnert 100% klassifisering. Men det skal helst ikke forekomme, for med mindre en har et stokastisk element vil en vekt på 1, hvor 1 er maksimums verdi, låse vektene som bidrar til klassifiseringen. Så en 100% klassifisering vil enten være resultat av en avrunding eller et klart tegn på et låst nevralt nettverk.

Forøvrig er det ikke vanskelig å få vedlegg tre til å gi en 100% klassifisering på en utnode. Det er bare å la den gå en stund, typisk noen hundre tusen epochs, og da vil normalt en utnode ligge på 100%. Men samtidig vil som regel en annen utnode bli liggende på 0,0% uansett inndata.

Det som gjenstår av essensiell testing av vedlegg 8.3 er å utsette den for et annet testsett etter endt læring. Her viser applikasjonen skuffende resultater, som vist i Tabell 13.

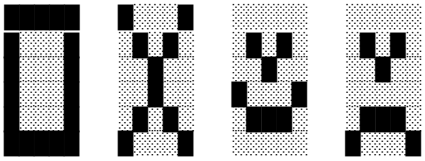
		Test 1				Test 2				Test 3			
		Classification by outnode				Classification by outnode				Classification by outnode			
		0	1	2	3	0	1	2	3	1	1	2	3
Dataset 1	Shape 0:	**87,8 %**	0,9 %	**11,3 %**	0,6 %	**99,9 %**	1,6 %	**0,1 %**	0,0 %	**98,4 %**	0,0 %	**1,1 %**	0,3 %
	Shape 1:	**12,3 %**	**94,8 %**	**0,0 %**	2,8 %	**2,8 %**	**98,3 %**	**0,1 %**	1,2 %	**0,1 %**	**97,9 %**	**0,2 %**	1,1 %
	Shape 2:	**0,5 %**	6,8 %	**90,1 %**	3,1 %	**0,3 %**	0,3 %	**98,8 %**	4,3 %	**0,9 %**	0,0 %	**98,4 %**	0,1 %
	Shape 3:	**0,5 %**	2,0 %	**1,1 %**	**99,1 %**	**0,3 %**	3,5 %	**4,2 %**	**96,2 %**	**0,4 %**	0,0 %	**1,6 %**	**98,2 %**
	MSE:				**0,0045**				**0,0005**				**0,0001**
Dataset 2	Shape 0:	***4,7 %***	72,4 %	**9,1 %**	0,1 %	***18,2 %***	68,6 %	**6,3 %**	0,0 %	***4,9 %***	0,0 %	**18,9 %**	5,7 %
	Shape 1:	**3,8 %**	*75,5 %*	**8,6 %**	0,2 %	**13,1 %**	*98,6 %*	**1,6 %**	0,0 %	**0,1 %**	*98,2 %*	**0,2 %**	1,0 %
	Shape 2:	***22,2 %***	0,0 %	***99,1 %***	5,1 %	**17,8 %**	0,1 %	***99,8 %***	0,0 %	**0,0 %**	0,0 %	***100,0 %***	3,0 %
	Shape 3:	**0,2 %**	41,2 %	**0,1 %**	*98,1 %*	**0,5 %**	3,1 %	**4,3 %**	*95,0 %*	**0,1 %**	98,3 %	**0,2 %**	*8,0 %*
	MSE:				**0,0045**				**0,0005**				**0,0001**

Tabell 13: Resultater fra klassifisering av et datasett nummer to etter opplæring av 8.2 med datasett en.

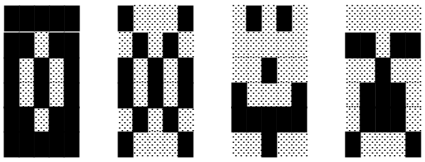
Det er altså først trent opp et nevralt nettverk på datasett en til MSE lik eller bedre enn 0,005, for så å kjøre dette mot datasett to.

Datasettene hadde følgende form:

Datasett 1:



Datasett 2:



En kuriositet her er at smil (”shape 2”) fra datasett to gjennomgående klassifiseres bedre enn det originale. Dessverre er det ikke langt unna at dette også stemmer for form nummer 2 (”shape 1”).

5.6.3 Egne eksempler sett i sammenheng med eksisterende forskning

Det fine med nevrale nettverk er at disse ikke trenger å levere ja eller nei svar. Den vanligste verdien et nevralt nettverk leverer som utdata er et tall mellom 0 og 1. Dette tallet vil som regel være angivende for hvor stor sannsynlighet det er for et korrekt svar, det vil si; ut data samsvarer med hvor godt inndata passer til lagrede strukturer (former) i det nevrale nettverket. Det medfører at det vil være trivielt å skille mellom for eksempel sikker kategorisering, usikker kategorisering og klar forkastelse. Altså kan man gjøre flere verktøy og/eller valg tilgjengelig for en bruker hvis svaret fra det nevrale nettverket er usikkert.

For uthenting av billedmateriale er NORB basen som inneholder bilder delt i fire forskjellige kategorier:

- norm-unif normalisert-uniformt datasett; bilde av et sentrert objekt fra forskjellige vinkler og med seks forskjellige lysnivåer og lik bakgrunn for alle bildene.

- jitt-unif jittered-uniformt datasett; uniform bakgrunn men med vilkårlige variasjoner i objektene
- jitt-text jittered-teksturert datasett; også forskjellige bakgrunner
- jitt-clutt jittered-cluttered datasett; andre objekter introdusert, svært varierende bakgrunner og mye variasjoner på objektene.

Bilder hentet fra Flickr vil typisk være av jittered-cluttered kvalitet. Men med enkle hjelpemidler som å la en bruker velge ut et område fra bildet, vil man fort og enkelt kunne forbedre dette til jittered-textured. Med hensyn til feilrate vil dette kunne forbedre klassifiseringen i størrelsesorden 39% (i henhold til [15] fra 16,7% feilrate ned til 10,6%). Går en ut ifra at [15] sine klassifikasjonsrater er oppnålige vil dette tilsvare en feil rate for brukerkontrollerte 96x96x8 (gråtone) bilder ved hjelp av Convolutional Net 100 på ned mot 16%. Huang Fu Jie og Yann LeCun ([123]) har fått denne feilraten helt ned mot 7,2% for jittered-cluttered datasettet ved kun å benytte CNN. Ved å kombinere SVM og CNN har de imidlertid fått feilraten helt ned mot 5,9%. Men en kombinasjon av SVM og CNN vanskeliggjør senere eksport av den nevrale nettverksmodellen. I tillegg medfører SVM store krav til maskin kapasitet. Så mye faktisk at [15] ikke så det plausibelt å kunne kategorisere bilder større enn 48x48x8.

Med ”state of the art” nevral nettverksmodell burde altså vedlegg 8.4 kunne komme ned mot en feilrate på 10% eller bedre. Fordelen med en slik applikasjon vil være den potensielt uendelige muligheten til videre opplæring. Uthentingen av bilder fra Flickr er ikke spesielt effektivt i 8.4, men mulighetene for forbedringer er mange. Et av de store problemene er tiden det tar å bla frem til passende bilder. I Tabell 10 har jeg altså gjort en liten undersøkelse på kvaliteten av bilder, og i henhold til mine funn vil noe sånt som halvparten av alle bilder i snitt måtte forkastes som opplæringsgrunnlag. For vedlegg 8.4 sin del resulterer det mye blaing i bilder og en god del venting på lasting av billedsett fra Flickr. Med threading av koden og oversikt over hele billedsett som i 8.3 kan slike problemer for overkommes. Andre problemer som automatisk siling av bilder, er nettopp det kapittel 3 danner et grunnlag for.

6 Oppsummering

Jeg vil her komme med en oppsummering av mine funn, både med hensyn til gjenomgått forskning og egne eksempler. Til slutt kommer jeg med min konklusjon over funnene jeg har gjort.

6.1.1 Basis for evaluering

Jeg har gått igjennom det jeg mener er relevant forskning og teknologi som kan danne et grunnlag for et system som kan trene opp nevrale nettverk og bruke disse utover originalt tilsiktet bruk.

Som datagrunnlag for slik opplæring har jeg foreslått å benytte sosiale nettsteder fordi slike nettsteder innebærer at brukerne har en kognitiv intensjon bak bruken. Hvis denne intensjonen er sammenfallende med intensjonen til det nevrale nettverket en ønsker å trene, vil datasett for treningen alltid inneholde relevant data fordi brukerne ser det som relevant i henhold til intensjonen. Forutsetter med andre ord at hyponymi ikke vil være et problem så lenge kognitiv intensjon er den samme.

Sosiale nettsteder er også egnet som grunnlag fordi de tilbyr en stadig tilgang på nye treningsdata. Med andre ord kan man intensjonelt fortette å trene et nevralt nettverk som baserer seg på et sosiale nettsted. Benyttes tilgjengelige datasett som MNist, NORB eller andre, har man en gitt mengde læring tilgjengelig og dermed et gitt maksimalt funksjonsnivå.

Videre har jeg sett på bruk av nevrale nettverk i større skala. Her har jeg tatt for meg to aspekter; prosessering og viderebruk av eksisterende løsninger.

6.1.2 Reservasjoner

Aller først må det sies at å skrive på norsk i denne sammenhengen ikke alltid har vært like lett. Det tas derfor et stort forbehold om mulige feil oversatte ord og utrykk, og av denne grunn har jeg også ofte inkludert eller rett og slett benyttet den engelske terminologien.

Grunnet mangfoldet av maskinlæringsalgoritmer har jeg valgt å konsentrere meg om kun nevrale nettverk. Nå er imidlertid omfanget av nevrale nettverk også så stort at det kan være vanskelig å ha oversikt over alle varianter. Jeg mener dog at jeg med grunnlag i den forskningen jeg har gått igjennom har funnet frem til de mest sentrale akademiske modellene, vel og merke til kategoriserings formål og primært med hensyn til billedmateriale.

Videre er nevrale nettverksmodeller tildels befestet i avansert matematikk. Jeg har etter beste evne forsøkt å sette meg inn i det matematiske grunnlaget såvidt

grundig at jeg forstår alle vesentlige aspekter i overordnede funksjoner og begrensninger.

At et system som den foreslåtte vil kunne fungere, sikrer selvfølgelig ikke at det vil fungere tilfredsstillende. Forskjellige mønstergjenkjenningsforsøk gjort med nevrale nettverk har oppnådd særdeles vidt forskjellige resultater avhengig av hvilke nettverksmodeller og læringsalgoritmer som har blitt brukt. *Gode* feilrate resultater kan variere fra 0,3% til 40%. Ikke minst vil kravene til et ferdig system variere avhengig av formål. I visse tilfeller kan 40% feilrate være akseptabelt så lenge alle ønskede elementer blir funnet, som ved gjenfinning av feriebilder. I andre tilfeller vil selv 1% feilrate karakteriseres som svært dårlig, som for eksempel OCR og fonem gjenkjenning. Jeg har ikke tatt stilling til hvor gode resultater et slikt system må kunne levere for å kalles en suksess, fordi jeg ikke har tatt stilling til spesifikt hva et slikt system kan brukes til. Jeg mener at det primært vil være fantasien som vil sette grensene for hvordan dette kan benyttes.

Sosiale nettsteder vil ikke nødvendigvis være den kvalitativt beste kilden til data. Det kan være vanskelig å skreddersy et effektivt sosialt nettsted med tanke på opplæring av nevrale nettverk fordi brukermasse ikke kan garanteres. Videre kan mitt forslag om å benytte eksisterende sosiale nettsteder gi et generelt dårlig kvalitativt grunnlag fordi intensjonene bak nettstedet og/eller brukerne ikke sammenfaller godt nok med intensjonen bak det nevrale nettverket. For eksempel har ikke alle brukere av Flickr.com den samme kognitive intensjonen. Altså vil det alltid være bilder i Flickr.com som vil stride mot en eventuell intensjon bak et nevralt nettverk.

På den annen side kan en godt tenke seg å lage et system hvor intensjon bak det nevrale nettverket er å bistå brukere av et sosialt nettsted. Da vil altså den kognitive intensjonen til det nevrale nettverket bli det brukerne har som intensjon, uavhengig av hvor sprikende dette måtte være. Dette er en kontrast til ekspert baserte systemer hvor løsningen alltid vil bære preg av den utviklende ekspertens intensjon.

6.2 Faktum

Det er et faktum at nevrale nettverk fungerer til en rekke oppgaver, deriblant mønstergjenkjenning, kategorisering og prediksjon. I henhold til nevral nettverks ekspert Maia Dimitrova er det også et faktum at nevrale nettverk kan trenes opp til en hvilken som helst grad av funksjonsnivå.

Det er videre et faktum at store mengder kategorisert eller kategoriserbart billedmateriale er fritt tilgjengelig ved hjelp av internett per i dag. Kvaliteten på, og nytten av kategoriseringen er derimot usikker. Også kvaliteten på tilgjengelig billedmateriale via internett er av svært varierende kvalitet.

Teknologi finnes tilgjengelig for å løse eventuelle kapasitetsmessige krav til nevrale nettverk med hensyn til prosesseringskraft og minnebruk. Det kan dog finnes begrensninger som per i dag vanskeliggjør en del nevrale nettverksmodeller og da spesielt ved bruk av store datasett som inndata kombinert med de mer avanserte tilgjengelige læringsteknikkene.

Løsninger for kapasitetsmessige krav som grid teknologi, kan også sette krav til funksjonalitet som vil gjøre en løsning ubrukelig.

Det er også et faktum at det finnes teknologi for flytting og dermed også distribuering av nevrale nettverksmodeller. Det kan dog være begrensninger på hvor komplekse nettverksmodellene kan være for å kunne benyttes direkte av andre systemer.

6.2.1 Kriterier

For å kunne lage et system slik beskrevet må det finnes en datakilde som kan levere datasett av en relativt gitt kvalitet. Hvor god denne kvaliteten må være vil variere, men det er vist at desto bedre kvalitet som oppnås, desto bedre vil kvaliteten på gjenkjenning/klassifisering være. Det vil alltid være et kritisk punkt med hensyn til kvalitet hvor det nevrale nettverket vil slutte å konvergere mot en gitt kategorisering. Dette kritiske punktet vil variere ikke bare av valg av nevral nettverksmodell og læringsmetode, men også av innstillingene til variabler benyttet i det nevrale nettverket slik vedlegg 8.1 klart viser.

Det er foreslått å kombinere taksonomi og tagge teknologi for å finne frem til best mulig kategorisering og uthenting av datasett. Hvis antall utparametre ikke kan gjøres dynamisk i henhold til den nevrale nettverksmodellen som velges, må det benyttes en taksonomi som grunnlag for det endelige valget for kategoriseringer. Det er altså svært ønskelig for et slikt foreslått system å ha dynamisk antall utparametre. Gjeldende forskning har dessverre per i dag ingen ideelle, lett tilgjengelige løsninger på dette problemet.

Det anbefales å velge en nevral nettverksstruktur som er kompatibel med PMML for å kunne dra nytte av et slikt system utover den initielt tiltenkte funksjonaliteten.

6.3 Teknisk evaluering

Jeg har laget en fungerende applikasjon som kombinerer bilder fra Flickr av gitte kategorier med et fungerende nevralt nettverk. Jeg har derimot ikke vist at en slik applikasjon kan fungere på en tilfredsstillende måte. Det er derimot vist gjennom presentert forskning at det finnes tilgjengelig teknologi som kan bøte på dette.

Det er mange forutsetninger for å kunne lage en slik god applikasjon. Desto bedre datagrunnlag, desto bedre applikasjon. Her har jeg foreslått å benytte

tagge teknologier som folksonomier og taggeskyer, samt kombinere dette med taksonomier som WordNet. Jeg har kun gjort dette på manuell basis, men WordNet kan aksesseres maskinelt og likeledes kan statistisk informasjon om tagger fra Flickr. Det er altså fullt mulig å maskinstyre alle aspekter ved tagge og taksonomi teknologier på miljøene som er brukt som eksempler. Mine eksperimenter har derimot vist at hel automatisering må gjøres med måtehold da dette relativt fort kan redusere datagrunnlaget til ingenting.

Det kan enkelt implementeres flere verktøy som kan heve kvaliteten på datagrunnlaget. Utsnittsfunksjonalitet for eksempel, kan enkelt gjøres lett tilgjengelig. Med det kan en bruker fort og enkelt heve kvaliteten på et bilde fra *jittered-cluttered* til *jittered-textured ([15])*. Det vil typisk medføre en klassifiseringsforbedring på over 50%.

Verktøy for automatisk balansering av lysstyrke er et annet eksempel som kan gjøre feilraten til et nevralt nettverk betydelig bedre. Ikke minst fordi slike nettverk normalt opererer med gråtoner.

6.3.1 Evaluering av teknisk konsept

Det er flere årsaker til at jeg har skrevet mesteparten av kildekoden min i C#:

- Det er relativt enkelt å operere på piksel nivå på bilder.
- Det er rimelig raskt.
- Jeg kjenner det relativt godt.
- Det støtter native objekt struktur.
- Det finnes masse eksempel kode, ikke minst fra [54], hvis man skulle stå fast.

Forøvrig skrev jeg for et halvt år siden et nevralt nettverk ganske likt vedlegg en i C# som heller ikke konvergerte. For å være helt sikker på å ikke kopiere eventuelle feil skrev jeg vedlegg 8.1. Grunnen til at jeg da valgte Lua som programmeringsspråk fordi jeg ville ha hastighet, enkelhet og portabilitet. På alle disse tre områdene er Lua, slik jeg ser det, uovertruffent. Vedlegg 8.1 kjører utmerket på min fem år gamle Palm. Piksel håndtering derimot er ikke like enkelt på PC. Det ville vært greit på PLua (PalmOS varianten), men da ville slitt med billedhenting fra Flickr som jeg allerede hadde gjort da jeg skrev 8.3. Til 8.3 hentet jeg forøvrig inspirasjon fra [54] og [143].

Den største ”feilen” ved C# koden min er at jeg ikke har benyttet threading. Det betyr blant annet at en testkjøring for å trene opp et nevralt nettverk til MSE <= 0,05 mot Flickr bilder ferdig kompilert applikasjon versjon av 8.4, har stått i over et døgn på en AMD Sempron 2600+ uten at jeg vet om den har kommet noen vei eller om den rett og slett har kræsjet.

Som nevnt har jeg kun benyttet online versjonen av WordNet. Jeg har altså gjort søk i Flickr med 8.3 og 8.4 ved hjelp av WordNet, kun på manuell basis.

Jeg har ikke sett noen grunn til å forsøke å automatisere dette før man har et fungerende nevralt nettverk, og mitt nevrale nettverk fungerer jo så langt kun får små og ukomplekse datasett.

Når det gjelder effektiviteten til koden er det en del og si. Det er viktig å legge merke til at et trenings kall består av mange ganger så mange operasjoner som et rent kategoriserings (feed) kall. I tillegg viser forskning og mine egne forsøk at en må gjøre mange tusen treningskall før et nevralt nettverk konvergerer. Et nevralt nettverk er altså tregt å trene opp - fra minutter til timer og mer - mens det som regel er svært raskt i bruk – hver prosessering kan som regel måles i noen få millisekunder.

Videre har jeg kun benyttet RawTags egenskapen til Flickr's billedkolleksjons-objekt. Og måtene jeg har benyttet denne på har kun vært ved å sjekke enten om en tag finnes eller om det er den eneste taggen som finnes. Her er det enkelt å tenke seg flere ting å teste ut, som å tillate en ekstra tag i tillegg til den det søkes etter. Men langt viktigere er det at jeg, mest grunnet mangel på tid og at jeg ikke følger mitt nevrale nettverk er klar for det, ikke benytter de mer avanserte tagge funksjonene i Flickr sitt api. Jeg har kun sett på Flickr sine presenterte folksonomier for manuelt å velge ut passende søkeord. Folksonomi sjekk burde selvfølgelig vært gjort programmatisk, men det må altså bli et annet sted enn her.

Jeg har heller ikke laget eksport funksjonalitet til PMML. Jeg har derimot søkt å holde meg innenfor spesifikasjonene til PMML. Det virkelige problemet med hensyn til PMML vil derimot oppstå hvis en implementerer CNN, PyraNet eller en annen avansert nevral nettverksstruktur. Jeg mener derfor at det vil være interessant å fortsette forskning på enkle nettverksstrukturer som MLP så lenge potensielle bruksområder ikke eksplisitt støtter mer avanserte strukturer.

6.3.2 Mulige anvendelser

Begrensningen til foreslåtte teknologi ligger først og fremst i tilgang til relevante datasett og eventuell kvalitet på dette. Jeg har i denne oppgaven fokusert på mønstergjenkjenning i billedmateriale. Men så lenge det finnes et datasett (inndata) og et tilgjengelig kategoriseringssett (utdata) kan prinsipielt nevrale nettverk benyttes. Om dette finnes på, eller er egnet for, sosiale nettsteder eller kan benyttes av PMML kompatible nevrale nettverksstrukturer er så sin sak.

Eksempler på mulig anvendelse kan være:

- **Objektgjenkjenning**
  Gjenkjenne objekter tilhørende en gitt kategori i bilder. Spesialtilfeller kan være ansiktsdeteksjon og ansiktsgjenkjenning. Video er et annet potensielt dataformat.
- **Musikk**
  Det finnes flere vitenskaplige og kommersielle eksempler på musikk

gjenkjenning og prediksjon allerede. Knyttet sammen med en tagger og/eller taksonomi kan et nevralt nettverk for eksempel gjenkjenne spesifikk sang, sjanger, eller predikere brukers musikksmak.

- **Fonemgjenkjenning**
  Fonemgjenkjenning knyttet sammen med ord og eventuelt semantisk taksonomi kan gjøre nevralt nettverk i stand til å gjenkjenne språk. Ved å tilknytte et sosialt aspekt kan en kanskje realisere en dialekt gjenkjenner og/eller oversetter, og muligens språkoversetter. Kombinert med for eksempel mobiltelefon vil datainnsamlingspotensialet være enormt.
- **Værvarsling**
  Det finnes mange som prater om været. La dem gjøre det på internett og få et nevralt nettverk til å lære av det.

6.3.3 Kjent teknologi

Nevrale nettverk er i dag kjent teknologi og i stadig større grad forstått teknologi. Tagging er foreløpig en lite utforsket teknologi selv om den i stadig større omfang blir tatt i bruk. Taksonomier er godt forstått og i regulær bruk til en rekke kategoriseringsformål. Dataminering er godt kjent, mye brukt i dag og under stadig utvikling. PMML derimot ser ut til å være en noe mindre kjent standard, men støttes av de største database og datamineringsprodusentene. Det spørs dog om det er mulig å representere de mest avanserte nevrale nettverksformene ved hjelp av PMML.

All teknologi som det foreslåtte systemet vil måtte bero på er altså etablerte, veldokumenterte og standardiserte. Det som tilsynelatende ikke har vært gjort enda er å kombinere alle disse teknologiene. At det er mulig å kombinere dem mener jeg å ha vist er plausibelt.

Det som gjenstår for å bevise at det er mulig å lage systemer slik jeg har foreslått er å gjennomføre atskillige eksperimenter med kombinasjoner av forskjellige sosiale nettsteder, taksonomier, nevrale nettverksmodeller og eksport til fortrinnsvis DBMS'er med tilgang til egnet type rådata.

6.4 Konklusjon

Jeg har sett på forskning innen bruk av nevrale nettverk til kategoriseringsoppgaver. Herunder har jeg sett på moderne strukturer, mulige effektiviseringsmetoder og mulige dynamiske utvidelser av slike nettverk. Jeg har også sett på forskning rundt folksonomier og taksonomier samt herunder tagging og taggeskyer. Videre har jeg sett på datamineringsteknologi med tanke på utvidet bruk av nevrale nettverk.

Min presentasjon av disse forskningsområdene viser at det er mulig å hente ut bilder fra internett via dataansamlinger på såkalte sosiale nettsteder og

kombinere dette med kategoriseringsteknologier – som folksonomier og/eller taksonomier – til et grunnlag for opplæring av kunstige nevrale nettverk. Det hensiktsmessige ved en slik teknologi vil være at det nevrale nettverket vil ta opp i seg den kognitive intensjonen til opplæringsgrunnlaget – altså brukernes kognitive oppfatning av dataene. Et nevralt nettverk opplært på denne måten vil altså dynamisk tilpasse seg brukerne, også over tid.

Videre mener jeg presentert forskning åpner for bruk av slik kategoriseringsteknologi utover kun bilder. Og ved hjelp av presentert, eksisterende teknologi kan slike nevrale nettverk, gitt visse forutsetninger, tas i bruk utover det originale funksjonsområdet.

Ved hjelp av egenprodusert programvare har jeg vist at det med Flickr.com som eksempel, er enkelt å bygge applikasjoner for uthenting av billedemateriale. Jeg har vist at det derimot ikke er trivielt å lage fungerende, gode nevrale nettverksløsninger. Men med henvisning til presentert forskning er det vist at det er mulig og at det finnes flere forskjellige tilfredsstillende fremgangsmåter. En vesentlig teknologi jeg ikke har funnet en god, eksisterende løsning på, og som nevral nettverksekspert Maia Dimitrova sier det har vært manglende ønske om å forske på; er dynamisk utvidelse av nevrale nettverk med hensyn til antall mulige kategoriseringer.

Problemstillingen er vist mulig å løse, men med dagens tilgjengelige teknologi vil et system for opplæring av nevrale nettverk ved hjelp av sosiale nettsteder måte være noe begrenset funksjonsmessig.

Det kan dog være et alternativ til dagens vanlige metode for opplæring av nevrale nettverk som normalt går ut på bruk av prefabrikerte datasett. Videre ser det ut til at fremskridende forskning vil kunne bøte på eventuelle funksjonelle begrensninger.

7 Referanser

[1] Phung, S. L. and A. Bouzerdoum
A pyramidal neural network for visual pattern recognition
IEEE transactions on neural networks **18**(2): 329-43, 2007

[2] Studer-Imwinkelried, Matthias
NeuroCarb - Artificial Neural Networks for NMR Structure Elucidation of Oligosaccharides
2006 June, Universität Basel

[3] LeCun, P. Yann
Retrieved 29-06-07, 2007, from http://yann.lecun.com

[4] Golder, S.A. and B.A. Huberman
The Structure of Collaborative Tagging Systems
Journal of Information Science, 2005. **32**(2): p. 198-208.

[5] Suchman, L.A.
Plans and situated actions - The problem of human-machine communication
Cambridge University Press – 1987

[6] Badawy, O. and A. Almotwaly
Combining neural network knowledge in a mobile collaborating multi-agent system
Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference on.

[7] Bengio, Y. and Y. LeCun
Scaling Learning Algorithms towards AI
MIT Press, Large-Scale Kernel Machines. 2007

[8] Collaborative (2007)
Artificial neural network
Last accessed: 11. Sept. 2007, from
http://en.wikipedia.org/wiki/Artificial_neural_network

[9] Collaborative (2007)
Paul Werbos
Last accessed: 11. Sept. 2007, from
http://en.wikipedia.org/wiki/Paul_Werbos

[10] Collaborative (2007)
Markov decision process
Last accessed: 11. Sept. 2007, from
http://en.wikipedia.org/wiki/Markov_decision_process

[11] Collaborative (2007)
Self-organizing map
Last accessed: 11. Sept. 2007, from http://en.wikipedia.org/wiki/Self-organizing_map

[12] Teuvo Kohonen (2001)
Helsinki University of Technology - Teuvo Kohonen
Last accessed: 12/9-07, from http://www.cis.hut.fi/teuvo

[13] Paul Werbos (2007)
Welcome to the Werbos World!
Lasted accessed: 29/10-07, from http://www.werbos.com

[14] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano and Kevin J. Lang (1989)
Phoneme Recognition Using Time-Delay Neural Networks
IEEE transactions on acoustics, speech, and signal processing 37(3): 328-339

[15] Yann LeCun, Fu Jie Huang and Léon Bottou (2004)
Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting
Proceedings of CVPR'04, IEEE Press

[16] Raul Rojas (1996)
Neural Networks - A Systematic Introduction
Springer-Verlag, Berlin

[17] *Nettbutikk eksempel*
Retrieved 13/9, 2007, from http://www.komplett.no

[18] *Neural Networks*
Retrieved 15/9, 2007, from
http://www.statsoft.com/textbook/stneunet.html

[19] Johannes B. Steffens, Hartwig Adam and Hartmut Neven (2005)
Method and apparatus for image analysis of a gabor-wavelet transformed image using a neural network
Last accessed: 1/7, from http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6,917,703.PN.&OS=PN/6,917,703&RS=PN/6,917,703

[20] Philipp Lenssen m.fl. (2007)
New: Google Image Search Categories
from http://blogoscoped.com/archive/2007-05-28-n84.html,
http://images.google.com,
http://www.nevenvision.com, and

http://googleblog.blogspot.com/2006/08/better-way-to-organize-photos.html

[21] Ziegler Cai-Nicolas, Simon Kai and Lausen Georg (2006)
Automatic computation of semantic proximity using taxonomic knowledge
Arlington, Virginia, USA, ACM Press

[22] Kumar Ravi, Novak Jasmine and Tomkins Andre (2006)
Structure and evolution of online social networks
Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. Philadelphia, PA, USA, ACM Press

[23] George Macgregor and Emma Mcculloch (2006)
Collaborative tagging as a knowledge organisation and resource discovery tool
(Emerald Group Publishing) Library Review 55(5)

[24] Said Mirza Pahlevi and Hiroyuki Kitagawa (2002)
Taxonomy-based adaptive Web search method
Proceedings International Conference on Information Technology: Coding and Computing: 320-5

[25] A. Hearst Marti (1999)
Untangling text data mining
College Park, Maryland, Association for Computational Linguistics

[26] Fayyad Usama and Uthurusamy Ramasamy (1996)
Data mining and knowledge discovery in databases
ACM Press. 39**:** 24-26

[27] Usama Fayyad, Gregory Piatetsky-Shapiro and Padhraic Smyth (1996)
From Data Mining to Knowledge Discovery in Databases
AI Magazine 17(3): 18

[28] Jongwan Kim, Jesung Ahn, Chong Sang Kim, Heeyeung Hwang and Seongwon Cho (1994)
A new competitive learning algorithm with dynamic output neuron generation
1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on Neural Networks 2: 6

[29] Kah-Key Sung, Tomaso Poggio
Example-based learning for view-based human face detection
MIT, desember 1994

[30] Henry A. Rowley, Shumeet Baluja, Takeo Kanade
Human Face Detection in Visual Scenes
Carnegie Mellon University, November 1996

[31] Ming-Hsuan Yang, David J. Kriegman, and Narendra Ahuja
Detecting faces in images: A survey.
IEEE transactions on pattern analysis and machine intelligence, vol. 24, no. 1, January 2002

[32] Alison Bosson, Gavin C. Cawley, Yi Chan, Richard Harvey
Non-Retrieval; Blocking Pornographic Images
July 2002

[33] Cameron Marlow, Mor Naaman, Danah Boyd, Marc Davis
HT06, tagging paper, taxonomy, Flickr, academic article, to read
ACM Press, August 2006

[34] Tim Berners-Lee, James Hendler, and Ora Lassila
The Semantic Web
Scientific American, mai 2001

[35] George A. Miller
WordNet: A Lexical Database for English
ACM Press, februar 1995

[36] Ian Foster
What is the Grid? A Three Point Checklist
Argonne National Laboratory & University of Chicago, juli 2002

[37] Michael S. Lew, Nicu Sebe, Chabane Djeraba, Ramesh Jain
Content-Based Multimedia Information Retrieval: State of the Art and Challenges
ACM Press, February 2006

[38] Leander Krammer, Erich Schikuta and Helmut Wanek
A grid based neural network execution service
ACTA Press, februar 2006

[39] Cihan H Dagli and Hsi-Chieh Lee
Impacts of data mining technology on product design and process planning
Chapman & Hall, 1997

[40] Rafael C. Gonzalez and Richard E. Woods
Digital Image Processing
Addison-Wesley, September 1993

[41] Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner
Gradient-Based Learning Applied to Document Recognition
IEEE, November 1998

[42] Tore Helstrup og Geir Kaufman
Kognitiv Psykologi
Fagbokforlaget, 2. opplag 2002

[43] James Frye, Rajagopal Ananthanarayanan, Dharmendra S. Modha
Towards Real-Time, Mouse-Scale Cortical Simulations
2007 February

[44] Peter Morville & Louis Rosenfeld
Information Architecture for the World Wide Web
O'REILLY, 3rd edition 2007 December

[45] *DMG*
Last retrieved 5/9, 2007, from http://www.dmg.org

[46] R. Vaillant, C. Monrocq and Y. Le Cun (1994)
Original approach for the localisation of objects in images
Vision, Image and Signal Processing, IEE Proceedings - 141(4): 245-250

[47] M. S. Lew (2000)
Next-generation Web searches for visual content
Computer 33(11): 46-53

[48] (2007) *Pittsburgh Pattern Recognition*
Last accessed: 16/9, from http://demo.pittpatt.com/

[50] M. Bahrami (1993)
Integration of knowledge acquired by different neural networks
Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on

[51] Paul J. Werbos
Backpropagation Through Time: What It Does and How to Do It
Proc. IEEE, Vol 78, No. 10, October 1990

[52] Remco C. Veltkamp and Mirela Tanase (2002)
Content-Based Image Retrieval Systems: A Survey
Department of Computing Science, Utrecht University

[53] Alfred Ultsch and Dieter Korus (1995)
Integration of Neural Networks with Knowledge-Based Systems
Proc. IEEE Int. Conf. Neural Networks, Perth/Australia, 1995

[54] (2007) *Flickr API*
Last accessed: 17/9, from http://flickr.com/services/api/

[55] (2007) *Facebook API*
Last accessed: 17/9, from
http://wiki.developers.facebook.com/index.php/API

[56] Trandafir Moisa, Dan Ontanu and Adrian Horia Dediu (2001)
Speech Synthesis Using Neural Networks Trained by an Evolutionary Algorithm
Computational Science - ICCS 2001: International Conference, San Francisco, CA, USA, May 28-30, 2001, Proceedings, Part II**:** 419

[57] Stanford University (2007)
Folding@home distributed computing
Last accessed: October 2007, from
http://folding.stanford.edu/English/Papers

[58] Berkeley University (2007)
SETI@home
Last accessed: October 2007, from http://setiathome.ssl.berkeley.edu/

[59] Collaborative (2007)
Grid computing
Last accessed: October 29, 2007, from
http://en.wikipedia.org/wiki/Grid_computing

[60] Amy M. Braverman (2004)
Father of the Grid
University of Chicago Magazine, April 2004, vol. 96, no. 4

[61] Ian Foster, Carl Kesselman, Jeffrey M. Nick and Steven Tuecke (2002)
Physiology of the Grid
Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[62] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes (1993)
Lua
The Pontifical Catholic University of Rio de Janeiro in Brazil, last accessed: October 2007 from http://www.lua.org/

[63] Syed Sibte Raza Abidi (1996)
Neural Networks and Child Language Development: Towards a `Conglomerate' Neural Network Simulation Architecture

[64] Burgener Robin (2006)
Artificial neural network guessing method and game
Patent nr.: SG126811

[65] Rouat Jean, Pinchevar Ramin, Loiselle Stephane, Tei Le Tan Thanh, Hai Anh Hoang, Lavoie Jean and Bergeron Jocelyn (2007)
Spatio-temporal pattern recognition using a spiking neural network and processing thereof on a portable and/or distributed computer
Patent nr.: WO2007071070

[66] *Boltzmann machine* (2007)
Last accessed: September 2007, from
http://en.wikipedia.org/wiki/Boltzmann_machine

[67] Du Chunhua Yang (2007)
Posture recognition method of human's face based on limited Boltzmann machine neural network
Patent nr.: CN1952953

[68] Chen Yan Li (2007)
Fuzzy-neural network method for classifying intelligently shape of occupational suits
Patent nr.: CN1952962

[69] Aoyanagi Toshio; Aoki Takaaki (2007)
SOM neural network learning control device
Patent nr.: JP2007052677

[70] Burmer Glenna C and Ciarcia Christopher A (2002)
Computer method for image pattern recognition in organic material
Patent nr.: WO02097714

[71] M. Goebel and L. Gruenwald (1999)
A survey of data mining and knowledge discovery software tools
ACM SIGKDD, June 1999

[72] Denise Ecklund (2002)
INF312
UiO

[73] Melissa Corley and Ted Knight (2007)
New Era of "Desktop Supercomputing" Made Possible with Parallel Processing Power on a Single Chip
Last accessed: October 4 2007, from
http://www.eng.umd.edu/media/pressreleases/pr062607_supercomputer.html

[74] (2007)
Parallel computing
Last accessed August 2007, from
http://en.wikipedia.org/wiki/Parallel_computing

[75] Dr. Frank Mueller and Jennifer Weston (2007)
NC State Engineer Creates First Academic Playstation 3 Computing Cluster
Last accessed August 2007, from
http://news.ncsu.edu/releases/2007/march/041.html

[76] Brand Estelle and Gerritsen Rob (1998)
Data Mining and Knowledge Discovery
Last accessed: February 2006, from
http://www.dbmsmag.com/9807m01.html

[77] (2005)
ISoft Alice 6.5
Last accessed: February 2006, from http://www.alice-soft.com/

[78] (2006)
Business Objects XI
Last accessed: February 2006, from http://www.businessobjects.com/

[79] (2007)
Neural Network Models
Last accessed: 5. okotober 2007, from http://www.dmg.org/v3-2/NeuralNetwork.html

[80] Stephen Poole (2001)
Next-generation data mining
Teradata Magazine Online

[81] Martin Halvey and Mark T. Keane (2007)
An Assessment of Tag Presentation Techniques
2007 (6. oktober)

[82] A. W. Rivadeneira, M. Gruen Daniel, J. Muller Michael and R. Millen David (2007)
Getting our head in the clouds: toward evaluation studies of tagclouds
San Jose, California, USA, ACM Press

[83] (2006)
SPSS, Clementine 10
Last accessed: February 2006, from http://www.spss.com/

[84] Yusef Hassan-Montero and Victor Herrero-Solana (2006)
Improving Tag-Clouds as Visual Information Retrieval Interfaces
Mérida, Spain

[85] Owen Kaser and Daniel Lemire (2007)
Tag-Cloud Drawing: Algorithms for Cloud Visualization

[86] A. Golder Scott and A. Huberman Bernardo (2006)
Usage patterns of collaborative tagging systems
Journal of Information Science 32(2): 198-208

[87] Mike Rote (2007)
Combined forces
Teradata Magazine June 2007

[88] Arlene Zaima and Robert Cooley (2007)
Rapid analytics
Teradata Magazine September 2007

[89] Tim Miller and Arlene Zaima (2005)
Keeping your customers
Teradata Magazine June 2005

[90] Hossein Farsi and Fereydoon Gobal (2007)
Artificial neural network simulator for supercapacitor performance prediction
Computational Materials Science 39(3): 678-683

[91] Hermann Haken (2007)
Towards a unifying model of neural net activity in the visual cortex
Cognitive Neurodynamics 1(1): 15-25

[92] Salford Systems (2006)
CART
Last accessed February 2006, from http://www.salford-systems.com/

[93] RuleQuest (2006)
Cubist 5
Last accessed February 2006, from http://www.rulequest.com/

[94] IBM (2006)
DB2 Intelligent Miner for Data
Last accessed February 2006, from http://www-306.ibm.com/software/data/iminer/fordata/index.html

[95] IBM (2006)
DB2 Mobility
Last accessed February 2006, from http://www.ibm.com

[96] D. BMiner technology inc (2006)
DBMiner 2.0
Last accessed February 2006, from http://www.dbminer.com/

[97] Bissantz (2006)
Delta Master 5
Last accessed February 2006, from http://www.bissantz.com/en/

[98] Insight Dimensional (2006)
DI Diver
Last accessed February 2006, from http://www.dimins.com/

[99] M. Weiss Sholom and Indurkhya Nitin (2006)
EDM - Enterprise Data-Miner
Last accessed February 2006, from http://www.data-miner.com/

[100] Sas (2006)
SAS® Enterprise Miner
Last accessed February 2006, from
http://www.sas.com/technologies/analytics/datamining/miner

[101] Fujitsu (2006)
GhostMiner
Last accessed February 2006, from
http://www.fqs.pl/business_intelligence/ghostminer

[102] Nasa Cosmic Department (2006)
NASA COSMIC. IND 1.0. University of Georgia
Last accessed February 2006, from
http://www.openchannelfoundation.org/projects/IND

[103] Angoss (2006)
KnowledgeStudio 5
Last accessed February 2006, from
http://www.angoss.com/products/studio.php

[104] Kxen (2006)
Analytic framework
Last accessed February 2006, from http://www.kxen.com/products/

[105] Microsoft (2006)
SQL Server 2005 Enterprise
Last accessed February 2006, from
http://www.microsoft.com/sql/prodinfo/features/compare-features.mspx

[106] Microsoft (2006)
SQL Server Mobile Edition 3.0
Last accessed February 2006, from http://msdn2.microsoft.com/en-us/sql/Aa336364.aspx

[107] MicroStrategy (2006)
MicroStrategy 8
Last accessed February 2006, from http://www.microstrategy.com/

[108] Insight Purple (2006)
MineSet 3.2
Last accessed February 2006, from
http://www.purpleinsight.com/products/index.shtml

[109] Kargupta Hillol, Park ByungHoon, Pittie Sweta, Liu Lei, Kushraj Deepali and Sarkar Kakali (2002)
MobiMine: Monitoring the Stock Market from a PDA
SIGKDD Explorations 3(2)

[110] Oracle (2006)
Data Mining 10g release 2
Last accessed February 2006, from http://www.oracle.com

[111] Quadrillion (2006)
Q-yield 4.2
Last accessed February 2006, from
http://www.quadrillion.com/qyield.shtm

[112] Sav (2006)
Zigzag
Last accessed February 2006, from http://savtechno.com/

[113] Aeronic (2006)
StockPack
Last accessed February 2006, from http://www.aeronic.com

[114] Azmy (2006)
SuperQuery Discovery
Last accessed February 2006, from http://www.azmy.com/

[115] L. L. C. Beiks (2006)
Wall Street Financial Assistant 3.1
Last accessed February 2006, from
http://ww.beiks.com/palm/ShowTitle.asp?TitleID=54

[116] Teradata (2006)
Warehouse Miner
Last accessed February 2006, from http://www.teradata.com

[117] R. D. Lawrence, G. S. Almasi, V. Kotlyar, M. S. Viveros and S. S. Duri (2002)
Personalization of Supermarket Product Recommendations

[118] University Princeton (2007)
WordNet 3.0
Last accessed October 2007, from
http://wordnet.princeton.edu/perl/webwn

[119] Netscape (2007)
dmoz - open directory project
Last accessed: October 2007, from http://www.dmoz.org/

[120] LeCun Yann and Cortes Corinna (2007)
The MNIST database of handwritten digits
Last accessed: October 2007, from http://yann.lecun.com/exdb/mnist/

[121] Collaborative (2007)
Spiking neural network

Last accessed: October 2007, from
http://en.wikipedia.org/wiki/Spiking_neural_network

[122] Jim Tørresen (1996)
Parallelization of Backpropagation Training for Feed-Forward Neural Networks
The University of Trondheim. Ph. D.**:** 257

[123] Huang Fu Jie and LeCun Yann (2006)
Large-Scale Learning with SVM and Convolutional Nets for Generic Object Categorization
IEEE Press

[124] Marc'Aurelio Ranzato, Poultney Christopher, Chopra Sumit and LeCun Yann (2006)
Efficient Learning of Sparse Representations with an Energy-Based Model

[125] Dong Jian-xiong (2007)
HeroSvm 2.1
Last accessed, from
http://www.cenparmi.concordia.ca/~jdong/HeroSvm.html

[126] Collaborative (2007)
Growing neural gas
Last accessed, from http://en.wikipedia.org/wiki/Growing_neural_gas

[127] Dong Jian-xiong, Ponson Dominique, Krzyzak Adam and Y. Suen Ching (2005)
Cursive word skew/slant corrections based on Radon transform
Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR2005)

[128] Dong Jian-xiong, Krzyzak Adam and Y. Suen Ching (2005)
Fast SVM Training Algorithm with Decomposition on Very Large Datasets
IEEE Trans. Pattern Analysis and Machine Intelligence 27(4): 603-618

[129] Dong Jian-xiong, Krzyzak Adam and Y. Suen Ching (2005)
An improved handwritten Chinese character recognition system using support vector machine
Pattern Recognition Letter 26(12): 1849-1856

[130] M. Ranzato, P. E. Taylor, J. M. House, R. C. Flagan, Y. LeCun and P. Perona (2007)
Automatic recognition of biological particles in microscopic images
Pattern Recognition Letters 22(1): 31-39 Advances in Neural Information Processing Systems (NIPS 2006)

[131] Dong Jian-Xiong, Krzyzak Adam and Y. Suen Ching (2002)
Local learning framework for handwritten character recognition
Engineering Applications of Artificial Intelligence 15(2002): 8

[132] S. Loos Hartmut and Fritzke Bernd (2007)
DemoGNG (Version 1.5)
Last accessed: October 2007, from http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/NG_2.html

[133] R. Tveter Donald (2007)
Chapter 2 The Backprop Algorithm
Last accessed, from http://www.dontveter.com/bpr/public2.html

[134] A. Montes de Oca Marco, Garrido Leonardo, Jos and L. Aguirre (2005)
An hybridization of an ant-based clustering algorithm with growing neural gas networks for classification tasks
Santa Fe, New Mexico, ACM Press

[135] Jos, Garc, Rodr a, guez, Angelopoulou Anastassia and Psarrou Alexandra (2006)
Growing Neural Gas (GNG): A Soft Competitive Learning Method for 2D Hand Modelling
Oxford University Press. E89-D**:** 2124-2131

[136] Kumar Chellapilla, Sidd Puri and Patrice Simard (2006)
High Performance Convolutional Neural Networks for Document Processing
La Baule (France)

[137] Andrés Calderón, Sergio Roa and Jorge Victorino (2003)
Handwritten Digit Recognition using Convolutional Neural Networks and Gabor Filters
Proceedings of the International Congress on Computational Intelligence (CIIC), National University of Colombia in Medellin

[138] M. Battista L. Jackel, H. Baird, J. Ben, J. Bromley, C. Burges, E. Cosatto, J. Denker, H. Graf, H. Katseff, Y. LeCun, C. Nohl, E. Sackinger, J. Shamilian, T. Shoemaker, C. Stenard, I. Strom, R. Ting, T. Wood and Zuraw C. (1995)
Neural-Net Applications in Character Recognition and Document Analysis
Neural-Net Applications in Telecommunications, Kluwer Academic Publishers

[139] Cheng-Lin Liu and Hiromichi Fujisawa (2005)
Classification and Learning for Character Recognition: Comparison of Methods and Remaining Problems
Seoul, Korea

[140] Rita L. Atkinson, Richard C. Atkinson, Edward E. Smith, Daryl J. Bem and Susan Nolen-Hoeksema (2000)
Hilgard's Introduction to Psychology
Harcourt Brace

[141] Collaborative (2007)
Feedforward neural network
Last accessed: October 2007, from
http://en.wikipedia.org/wiki/Feedforward_neural_network

[142] Collaborative (2005-2007)
GFLOPS capabilities in selected CPU's and GPU's
Last accessed: October 2007, from
http://www.windowsfordevices.com/news/NS3988467635.html
http://en.wikipedia.org/wiki/Cell_microprocessor
http://www.realworldtech.com/page.cfm?ArticleID=RWT072405191325&p=2
http://home2.btconnect.com/hgi/ps3/ps3-official.html
http://itpro.no/omtaler.php?op=Maskinvare&id=1653
http://movementarian.com/2006/08/18/flops-mips-watts-and-the-human-brain
http://www.tomshardware.com/2007/07/16/cpu_charts_2007/page36.html
http://www.computerbase.de/artikel/hardware/grafikkarten/2006/test_nvidia_geforce_8800_gtx/2
http://en.wikipedia.org/wiki/AMD_Stream_Processor

[143] Kristof Leroux (2007)
Tickr: Create your own Flickr diashow.
Last accessed: October 2007, from
http://www.codeproject.com/useritems/Tickr.asp

[144] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986)
Learning internal representations by error propagation.
In Rumelhart, D. E. and McClelland, J. L., editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations Volume 1: Foundations, MIT Press, Cambridge, MA.

8 Vedlegg

Følgende fire vedlegg er applikasjoner jeg har laget for å kunne eksemplifisere vesentlige poeng i oppgaven.

8.1 Et nevralt nettverks eksempel i Lua

Dette var mitt andre forsøk på å lage et fungerende nevralt nettverk. Regnet med å få dette til å virke siden jeg hadde det matematiske grunnlaget til å sjekke alle verdier rundt om i nettverket for praktisk talt første epoch. Ved å sette vektene til gitte verdier (linje 34 til 41) fungerer for så vidt nettverket korrekt som XOR ALU. Ved trening konvergerer det imidlertid ikke mot korrekt verdi, høyst sannsynlig grunnet overlæring. Følger altså de matematiske formlene gitt i [40] og [133].

```
-- MLP_Neural_Network.lua
-- Note: Lua only simulates objects through its tables.
--       To learn more about lua visit www.lua.org

--********************
--* Connection class *
--********************
Con = {}

function Con.get (c)
  return (c.w * c.f:get())
end

function Con.backprop (c, a, d)
  if ((c.w + a) < 100) and ((c.w + a) > -100) then
    c.w = c.w + a
  end
  c.f:backprop(a, d)
end

function Con.set (c, weight)
  c.w = weight
end

function Con:new (weight, fromNeuron, toNeuron)
  local c = {
    w = weight,
    f = fromNeuron,
    t = toNeuron -- for more debug options only
  }

  -- to show the network works - initialize to known values
--[[  if (c.f.id == 1) and (c.t.id == 6) then c.w = -4.95
```

```
  elseif (c.f.id == 1) and (c.t.id == 4) then c.w = 7.1
  elseif (c.f.id == 3) and (c.t.id == 4) then c.w = -2.76
  elseif (c.f.id == 2) and (c.t.id == 4) then c.w = 7.1
  elseif (c.f.id == 2) and (c.t.id == 6) then c.w = -4.95
  elseif (c.f.id == 4) and (c.t.id == 6) then c.w = 10.9
  elseif (c.f.id == 5) and (c.t.id == 6) then c.w = -3.29
  end
]]
  -- The following line is for propagating "class" functions to the
new "object"
  setmetatable(c, {__index = self})
  return c
 end

--****************
--* Neuron class *
--****************
Neuron = {}

function Neuron.set (n, value)
  n.v = value
end

function Neuron.get (n)

  if table.getn(n.cons) > 0 then
    -- Summation of connection values before sigmoid
    local sum = 0
    for i = 1, table.getn(n.cons) do
      sum = sum + n.cons[i]:get()
    end

    -- logistic - Dimitrova/Tveter edition
    s = 1 / (1 + (math.exp(-sum)))
    -- tanh versjon. Anbefalt aktiverings formel.
    --s = math.asin(sum) / math.acos(sum)

    n.v = s
    return s
  else
    -- No connections means this is a retinal neuron (first layer
input)
    return n.v
  end
end

function Neuron.backprop (n, r, d, output)
  local eta = 0.5
  local delta = 0
  local beta = 0

  if output then    -- we're at the output neuron
    delta = (r - n.v) * n.v * (1 - n.v)
    for i = 1, table.getn(n.cons) do
      beta = eta * delta * n.cons[i].f.v
      n.cons[i]:backprop(beta, delta)
    end
```

```
  else                              -- hidden layers
    delta = n.v * (1 - n.v) * d * r
    for i = 1, table.getn(n.cons) do
      beta = eta * delta * n.cons[i].f.v
      n.cons[i]:backprop(beta, delta)
    end
  end
end

function Neuron:new (prevLayer, bias)
  -- Neurons contains a value which is actually just used by
  --  retinal neurons since all other neurons will be asked
  --  for values recursively by output. The v field is still
  --  updated for debuging reasons though.
  -- The id field is for debuging and future PMML use only.
  -- cons is the table holding connection "objects"
  idCounter = idCounter + 1
  local n = {
    v = 0,
    id = idCounter,
    cons = {}
  }
  local tmpL = prevLayer

  if prevLayer then
    for i = 1, table.getn(prevLayer.neurons) do
      -- Gives random weights to neew connections to start with
      --table.insert(n.cons, Con:new((1-(math.random(20)/10)),
      --   prevLayer.neurons[i]))
      -- Dimitrova/Tveter edition
      -- the reference to this neuron is to be able to initialize
      --   weights to specific values for known nn structures
      table.insert(n.cons, Con:new(0, prevLayer.neurons[i], n))

      -- Hopfield: The bias neuron will always be the last so we
      --           skip that here
      while tmpL.pLayer do
        tmpL = tmpL.pLayer
        for j = 1, (table.getn(tmpL.neurons) - 1) do
   table.insert(n.cons, Con:new(0, tmpL.neurons[j], n))
  end
      end
    end
  else
    cons = nil
    if bias then n.v = bias end
  end

  setmetatable (n, {__index = self})
  return n
end

--***************
--* Layer class *
--***************
-- Just a container class for neurons
```

```
Layer = {}

function Layer:new (nNeurons, prevLayer, bias)
  local l = {
    neurons = {},
    pLayer = prevLayer
  }

  for i = 1, nNeurons do
    table.insert(l.neurons, Neuron:new(prevLayer))
  end
  -- The bias is only for the next level
  if bias then table.insert(l.neurons, Neuron:new(nil, bias)) end

  setmetatable(l, {__index = self})
  return l
end

--************************
--* Neural network class *
--************************
-- The container class for layers. Here though are all the
initialization functions

NN = {}

function NN.overview(nn)
-- Overview is just for show(ing whats created)

  print ("The neural network structure contains:")
  count1 = 0
  for i = 1, table.getn(nn.layers) do
    count2 = 0
    for j = 1, table.getn(nn.layers[i].neurons) do
      count2 = count2 + table.getn(nn.layers[i].neurons[j].cons)
    end
    print ("Layer " .. i .. ": " .. table.getn(nn.layers[i].neurons)
.. " neurons and " .. count2 .. " connections")
    count1 = count1 + count2
  end
  print ("The total amount of connections and hence trainable
parameter is " .. count1)
end

function NN.feed(nn, input, result)
-- Feed is the function that does the recursive call through the
--   neural nettwork
-- First it sets first layer neurons to the values given in the input
--   array
-- Then it askes for output(s) by the neurons in the last layer which
--   does their recursive calls
-- If a result is given it back propagates the network with the
--   output and the wanted result

  output = {}
```

```
  -- +1 in the following because of bias neutron
  if table.getn(input)+1 ~= table.getn(nn.layers[1].neurons) then
    print ("The input size doesn't equal number of input neurons! \n"
           .."Don't blame me for crashing!")
  end

  -- Feeds input data to first layer
  for i = 1, table.getn(input) do
    nn.layers[1].neurons[i]:set(input[i])
  end

  -- Get outdata
  tmp_n = nn.layers[table.getn(nn.layers)].neurons
  for i = 1, table.getn(tmp_n) do
    val = tmp_n[i]:get()
    table.insert(output, val)
  end

  -- Check for (predefined) result. If so; goto training mode.
  if result then

    -- Compute mean square error
    for i = 1, table.getn(result) do
      mse = mse + (result[i] - output[i])^2
    end
    mse = mse / table.getn(result)

    -- Backpropagate
    for i = 1, table.getn(result) do
      tmp_n[i]:backprop(result[i], 0, 1)
    end

  else

    -- No (predefined) result was given; ie. not training. Report
    --   findings to user.
    for i = 1, table.getn(output) do
      print ("Result "..i..": "..string.format("%.2f", output[i]))
    end
    return output
  end
end

function NN:new (struct)
  -- Creates the neural network according to struct which denotes how
  --   many neurons each layer should contain. All neurons makes
  --   connections to all neurons in the previous layer.
  -- Note: This will only result in MLP's. Ie. as is - this program
  --   can not create advanced structures like CNN or PyraNet.
  -- Dimitrova/Tveter edition: Hopfield version with bias neuron for
  --   layer bias neuron not included in Hopfield - only connection
  --   from next layer

  local nn = {
    layers = {}
  }
```

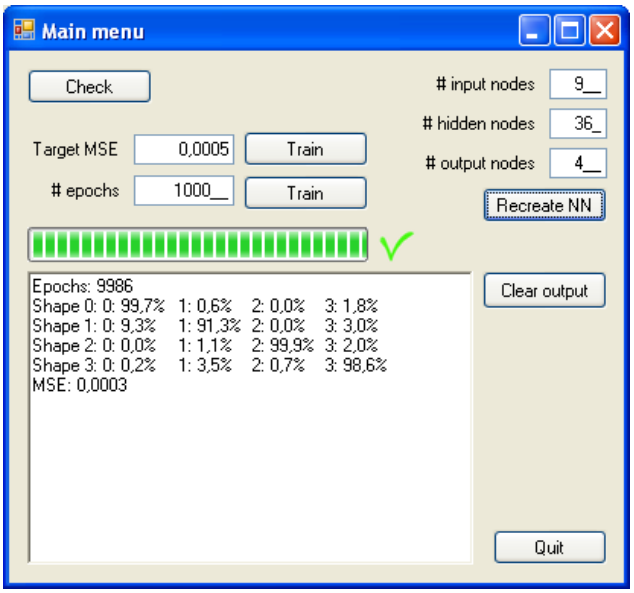
```
  local nl = table.getn(struct)

  -- Input layer
  table.insert(nn.layers, Layer:new(struct[1], nil, 1))

  -- Hidden layer(s)
  for i = 2, (nl - 1) do
    table.insert(nn.layers, Layer:new(struct[i], nn.layers[i-1], 1))
  end

  -- Output layer
  table.insert(nn.layers, Layer:new(struct[nl], nn.layers[nl-1]))

  setmetatable(nn, {__index = self})
  return nn
end

--*******************************
--* Class definitions finnished *
--*******************************

function show_weights()
  -- Writes all weights to screen sorted by layers and neurons.

  for i = 1, table.getn(net.layers) do
    print ("Layer " .. i)
    for j = 1, table.getn(net.layers[i].neurons) do
      if table.getn(net.layers[i].neurons[j].cons) > 0 then
        io.write (" Neuron "..j..": ")
        for k = 1, table.getn(net.layers[i].neurons[j].cons) do
          t = net.layers[i].neurons[j].cons[k].w
    if t then io.write(string.format("%.1f", t) .. " | ") end
        end
  print ()
      else
        --io.write (" Retinal")
      end
    end
    --print()
  end
end

function show_neurons()
  -- Writes all neurons values and ids to screen sorted by layers and
  --    neurons.

  for i = 1, table.getn(net.layers) do
    print ("Layer " .. i)
    for j = 1, table.getn(net.layers[i].neurons) do
      s = net.layers[i].neurons[j].id
      t = net.layers[i].neurons[j].v
      if t then io.write (s..":"..string.format("%.2f", t) .. "|")
end
    end
    print ()
  end
end
```

```
function feed(counter, s)
  -- counters for feedback
  local ic = {}
  for j = 1, table.getn(shapes) do ic[j] = 0 end
  local c = 0
  local result = {}
  local r = s
  if not s then r = 0 end

  for i=1, counter do

    if not s then
      r = r + 1
      if r == 5 then r = 1 end
    end

    result = {fasit[r]}

    if show_feed == 1 then show_shape(shapes[r]) end

    net:feed(shapes[r], result)
    ic[r] = ic[r] + 1

    -- and just to see that it's not dead
    c=c+1 if c >= life_interval then io.write (".") c=0 end
  end
  print ()
  print ("Fed the network with:")
  for i = 1, table.getn(shapes) do
    print ("Shape "..i..": "..ic[i])
  end
end

math.randomseed(os.date("%d%H%M%S"))

shapes = {}
-- The XOR dataset
table.insert(shapes, {1,0})
table.insert(shapes, {0,0})
table.insert(shapes, {0,1})
table.insert(shapes, {1,1})
fasit = {1,0,1,0}
nnStruct = {2,1,1}

idCounter = 0
mse = 0
net = NN:new(nnStruct)
net:overview()
show_feed = 0
life_interval = 1000

--**************
--* Event loop *
--**************
a = "1"
while a ~= "q" do
```

```
io.write ("\nCommand (? for help): ")
a = io.read ("*line")

if a == "sw" then
  show_weights()

elseif a == "sid" then
  show_neurons()

elseif a == "sf" then
  show_feed = math.abs(show_feed - 1)
  if show_feed == 1 then print ("Will show drawings")
  else print ("Will NOT show drawings") end

elseif string.sub(a, 1, 1) == "s" then
  n = tonumber(string.sub(a, 2, string.len(a)))
  if n then
    show_shape(shapes[n])
  end

elseif string.sub(a, 1, 2) == "ts" then
  s = tonumber(string.sub(a, 3, 3))
  n = tonumber(string.sub(a, 5, string.len(a)))
  if n then
    feed(n, s)
  end

elseif string.sub(a, 1, 1) == "t" then
  n = tonumber(string.sub(a, 2, string.len(a)))
  if n then
    feed(n)
  end

elseif string.sub(a, 1, 1) == "r" then
  n = tonumber(string.sub(a, 2, string.len(a)))
  if n then
    net:feed(shapes[n])
  end

elseif a == "?" then
  print ("\nsw    Shows all weights - ie. in neural network")
  print ("sid   Shows all neurons ids and values")
  print ("sX    Shows the shape X (1=happy, 2=sad, 3=square, \n"..
         "                         4=circle, 5=noise")
  print ("rX    Try to recognise shape X")
  print ("tsX:N Train the shape X for N iterations")
  print ("tX    where X is a number of 1 or more. The \n"..
         "       training algorithm randomly picks a shape \n"..
         "       for each iteration.")
  print ("sf    Show 'drawings' of feed during training - or not")
  print ("?     Shows this helpful message\n")
  print ("\nWritten by Steinar Hamdahl as part of a master "..
         "thesis\nFeel free to use this as you wish, but "..
         "please report to steinar@hamdahl.no if you get it \n"..
         "to work!!!\n\n")
end
```

8.2 Et fungerende MLP i C#

I dette programmet har jeg laget et fungerende forovermatet nevralt nettverk. Det benytter bakover propagering som treningsmetodikk. Treningen har jeg basert på array basert C kilde kode fra Maya Dimitrova. Hun har i sin tur basert seg på [144] sin veiledning for bakover propagerings algoritme.



Figur 6: GUI etter testkjøring med først Target MSE og deretter 1000 epochs.

Første kode gjengivelse er selve nevrale nettverksklassen, mens den andre kode biten er hvordan programmet benytter denne klassen. Utelatt kode er triviell kode for generering av GUI elementer og liknende.

8.2.1 Nevral nettverksklasse

```
public class Weight {
   private double w;
   private double c;
   private double n;
   private Node f;

   public Weight (Node from, Node to, Random r) {
      w = (r.NextDouble() - 0.5) / 3;
      f = from;
   }

   public double get () {
      return f.get() * w;
   }
   public double get_only_w () {
      return w;
   }
   public void add (double value) {
      w += value;
   }
   public Node from {
      get { return f; }
   }
   public double change {
      get { return c; }
      set { c = value; }
```

```
   }
   public double new_w {
      get { return n; }
      set { n = value; }
   }
} // of Weight


public class Node {
   protected int id;            // for PMML support
   protected double v;          // activation
   protected double net;
   protected double bias;
   protected double delta;
   protected double error;
   protected double bias_delta;
   protected double bed;
   protected Weight[] weights;
   protected double Alfa = 0.1;
   protected double Epsilon = 0.1;
   protected double Bepsilon = 0.1;

   public Node (Random r, int id, Layer prev_layer) {
      Node[] tmp_n;
      this.id = id++;
      this.bias = 0.12;
      if (prev_layer != null) {
         weights = new Weight[prev_layer.nodes.Length];
         for (int i = 0; i < prev_layer.nodes.Length; i++) {
            tmp_n = prev_layer.nodes;
            weights[i] = new Weight(tmp_n[i], this, r);
         }
      }
   }

   public Node () { }         // A "null" reference

   public virtual double get () { return v; }
   public double err {
      get { return error; }
      set { error = value; }
   }

   // backpropagation
   public void compute_weight_change () {
      foreach (Weight w in weights) {
         w.change = delta * w.from.get();
      }
      bed += delta;
   }
   public void update_weights () {
      foreach (Weight w in weights) {
         w.new_w = Alfa * w.new_w + Epsilon * w.change;
         w.add(w.new_w);
         w.change = 0;
      }
      bias_delta = Bepsilon * bed + Alfa * bias_delta;
```

```
      bias += bias_delta;
      bed = 0;
    }
  }  // of Node base class

  public class Retinal_Node:Node {
    public Retinal_Node (Random r, int id, Layer prev_layer)
        : base(r, id, prev_layer) { }
    public override double get () {
      return v;
    }
    public void set (double value) {
      this.v = value;
    }
  }

  public class Hidden_Node:Node {
    public Hidden_Node (Random r, int id, Layer prev_layer)
        : base(r, id, prev_layer) { }
    public override double get () {
      // tanh
      net = bias;
      foreach (Weight w in weights) {
        net += w.get();
      }
      v = Math.Tanh(-net);
      return v;
    }

    // Backpropagation
    public void compute_delta () {
      delta = error * (v * (1 - v));
    }
  }

  public class Out_Node:Node {
    public Out_Node (Random r, int id, Layer prev_layer)
        : base(r, id, prev_layer) { }
    public override double get () {
      // sigmoid
      net = bias;
      foreach (Weight w in weights) {
        net += w.get();
      }
      v = 1 / (1 + Math.Exp(-net));
      return v;
    }

    // Backpropagation
    public double compute_error (double target) {
      error = target - v;
      delta = error * (v * (1 - v));

      return error;
    }
    public void compute_sum_delta_error () {
      foreach (Weight w in weights) {
```

```
        w.from.err += w.get_only_w() * delta;
      }
    }
}


public class Layer {
   protected Node[] n;

   public Layer (int no_nodes, int id, Layer prev_layer,
                 bool last, Random r) {
      n = new Node[no_nodes];
      for (int i = 0; i < no_nodes; i++) {
         if (prev_layer == null) {
            n[i] = new Retinal_Node(r, id++, prev_layer);
         } else if (last) {
            n[i] = new Out_Node(r, id++, prev_layer);
         } else {
            n[i] = new Hidden_Node(r, id++, prev_layer);
         }
      }
   }

   public Node[] nodes {
      get { return n; }
   }
} // of Layer


public class NN {
   protected Layer[] l;
   Layer prev_layer = null;
   bool last;
   int counter = 0;
   Random r = new Random(unchecked((int)DateTime.Now.Ticks));

   public Layer[] layers {
      get { return l; }
   }

   public NN (int[] structure) {
      l = new Layer[structure.Length];
      for (int i = 0; i < structure.Length; i++) {
         if (i == structure.Length - 1) { last = true; }
         l[i] = new Layer(structure[i], counter, prev_layer,
                          last, r);
         prev_layer = l[i];
         counter += l[i].nodes.Length;
      }
   }

   public double[] activate () {
      double[] output = new double[l[l.Length - 1].nodes.Length];
      int h;

      // compute hidden layers
      for (h = 1; h < l.Length - 2; h++) {
```

```
        for (int i = 0; i < l[h].nodes.Length; i++) {
          l[h].nodes[i].get();
        }
      }

      // compute output layer
      h = l.Length - 1;
      for (int i = 0; i < l[h].nodes.Length; i++) {
        output[i] = l[h].nodes[i].get();
      }

      return output;
    }

    public double update (int[] target) {
      double[] result = activate();
      double SSE = 0;
      double err = 0;
      int i;
      int no_out = l[l.Length - 1].nodes.Length;
      int no_hidden_l = l.Length - 2;

      // compute output delta
      for (i = 0; i < l[2].nodes.Length; i++) {
        Out_Node n = (Out_Node)l[2].nodes[i];
        err = n.compute_error(target[i]);
        SSE += (err * err) / no_out;
      }
      foreach (Out_Node n in l[2].nodes) {
        n.compute_sum_delta_error();
      }
      foreach (Hidden_Node n in l[1].nodes) {
        n.compute_delta();
      }

      // compute changes
      foreach (Out_Node n in l[2].nodes) {
        n.compute_weight_change();
      }
      foreach (Hidden_Node n in l[1].nodes) {
        n.compute_weight_change();
      }

      // update changes
      foreach (Hidden_Node n in l[1].nodes) {
        n.update_weights();
      }
      foreach (Out_Node n in l[2].nodes) {
        n.update_weights();
      }

      return SSE;
    }
  } // of NN
```

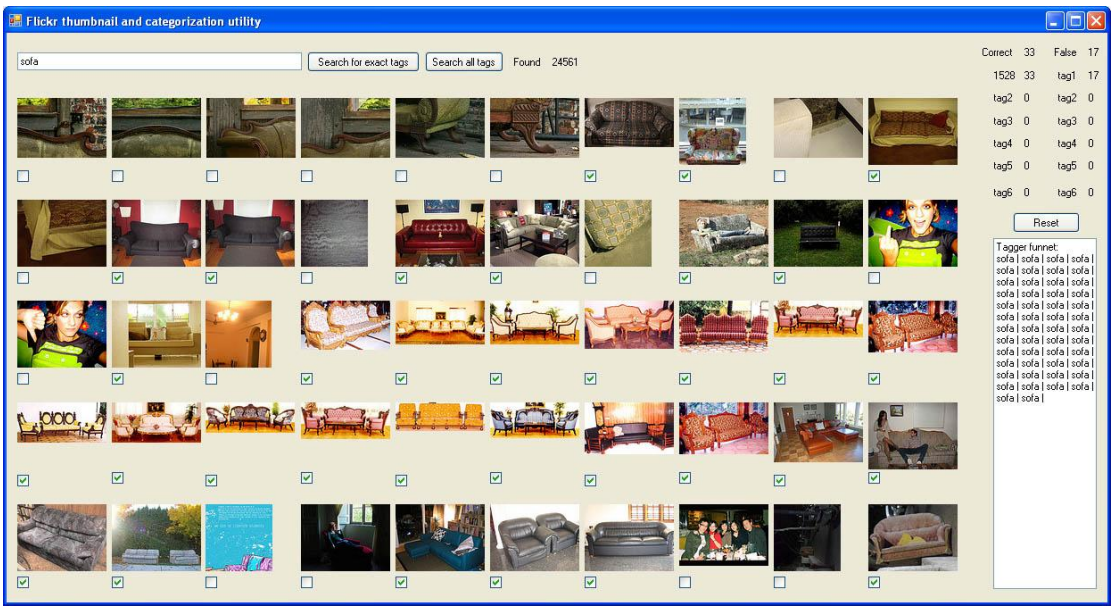
8.2.2 Styringsprogrammet / GUI

```
public partial class frmMenu:Form {

   NN nn;
   int[,] input = { { 1, 1, 1, 1, 0, 1, 1, 1, 1 },
                    { 1, 0, 1, 0, 1, 0, 1, 0, 1 },
                    { 0, 1, 0, 1, 1, 1, 0, 1, 0 },
                    { 0, 1, 0, 1, 0, 1, 0, 1, 0 } };
   int[,] target = { { 1, 0, 0, 0 },
                     { 0, 1, 0, 0 },
                     { 0, 0, 1, 0 },
                     { 0, 0, 0, 1 } };
   double MSE = 0;

   public frmMenu () {
      InitializeComponent();
   }

   private void frmMenu_Load (object sender, EventArgs e) {
      int[] structure = { Convert.ToInt16(txtInputNodes.Text),
        Convert.ToInt16(txtHidden1.Text),
        Convert.ToInt16(txtOutputNodes.Text) };
      nn = new NN(structure);
   }

   private void btnCreateNN_Click (object sender, EventArgs e) {
      int[] structure = { Convert.ToInt16(txtInputNodes.Text),
        Convert.ToInt16(txtHidden1.Text),
        Convert.ToInt16(txtOutputNodes.Text) };
      nn = new NN(structure);
   }

   private void btnQuit_Click (object sender, EventArgs e) {
      this.Close();
   }

   private void feed (int i) {
      Retinal_Node tmp_n;
      for (int j = 0; j < 9; j++) {
        tmp_n = (Retinal_Node)nn.layers[0].nodes[j];
        tmp_n.set(input[i, j]);
      }
   }

   private void btnCheck_Click (object sender, EventArgs e) {
      double[] result = new double[8];

      for (int i = 0; i < 4; i++) {
        feed(i);
        txtOut.Text += "Shape " + i.ToString() + ": ";
        result = nn.activate();
        for (int j = 0; j < 4; j++) {
          txtOut.Text += j + ": " +
         Convert.ToDouble(result[j] * 100).ToString("##0.0") +
         "%\t";
        }
```

```
            txtOut.Text += "\n";
        }
        txtOut.Text += "MSE: " + MSE.ToString("0.0000") + "\n\n";
    }

    private void btnTrain_Click (object sender, EventArgs e) {
        int[] t = new int[4];
        double part_MSE;

        picOK.Visible = false;
        progress.Maximum = Convert.ToInt16(txtEpochs.Text);
        progress.Value = 0;
        for (int epochs = 0; epochs <
            Convert.ToInt16(txtEpochs.Text); epochs++) {
            part_MSE = 0;
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) { t[j] = target[i, j]; }
                feed(i);
                part_MSE += nn.update(t);
            }
            progress.Value += 1;
            MSE = part_MSE / 4;
        }
        picOK.Visible = true;
    }

    private void btnTargetMSE_Click (object sender, EventArgs e) {
        int[] t = new int[4];
        int counter = 0;
        double part_MSE;

        MSE = 10;
        picOK.Visible = false;
        progress.Maximum = 1000;
        progress.Value = 1000;
        while (MSE > Convert.ToDouble(txtTargetMSE.Text)) {
            part_MSE = 0;
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) { t[j] = target[i, j]; }
                feed(i);
                part_MSE += nn.update(t);
            }
            MSE = part_MSE / 4;
            progress.Value = Convert.ToInt16((MSE * 1000));
            counter++;
        }
        picOK.Visible = true;
        txtOut.Text += "Epochs: " + counter.ToString() + "\n";
    }

    private void cmdClear_Click (object sender, EventArgs e) {
        txtOut.Text = "";
    }

}
```

8.3 Et C# eksempel på uthenting av bilder fra Flickr – hvor enkelt det er.

Her har jeg laget et program for å teste billeduthenting fra Flickr, spesielt med hensyn på å få en oversikt over kvaliteten på returnerte bilder. De linjene som direkte har noe med hentingen av bilder å gjøre har fått litt større skrift enn resten. Alle komponenter som har med visning av Flickr bilder og informasjon å gjøre har standard navn gitt av C#. Koden er ikke optimalisert for hastighet eller feil håndtering. Men det elementet som tar mest tid er uansett selve oppdateringene av resultatsettene fra Flickr, og den koden ligger altså i ekstern .dll som er hentet fra www.flickr.com. Erfaringsmessig kan det gå fortere å benytte web grensesnittet til Flickr, men dette gir ikke samme kontroll over tag håndteringen.



Figur 7: Et program for henting av bilder fra Flickr og påfølgende enkel statistikk generering av funn. Tiltenkt sortering av bilder etter ”kvalitet” slik beskrevet i [15].

Når det gjelder selve koding burde kontrollene vært i et programstyrt array, men for å spare tid har jeg veldig enkelt og raskt kopiert opp kontroll forekomster til jeg kom til 40 stk. Et positivt resultat av dette er at leser ikke trenger kontroll genereringskoden som ikke er gjengitt her, fordi denne er helt standard kode generert av C# siden jeg ikke har byttet navn fra deafult på noen komponenter som burde vært i array. Men det er altså lett og gjøre denne koden adskillig mer ”elegant”.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;

using FlickrNet;    // Fritt nedlastbar fra Flickr.com


//using System.Threading;


namespace FlickrGet {
   public partial class MainSearch : Form {
      Flickr flickr;

      public MainSearch() {
         InitializeComponent();
         string apikey = "[redacted]";
         flickr = new Flickr(apikey);
      }


      private Image GetImage(string sURL) {
         Stream str = null;
         HttpWebRequest wReq =
(HttpWebRequest)WebRequest.Create(sURL);
         HttpWebResponse wRes =
(HttpWebResponse)(wReq).GetResponse();
         str = wRes.GetResponseStream();

         return Image.FromStream(str);
      }


      private void updateFindings(int i, int j) {
         int t;

         lblCorrect.Text = Convert.ToString(Convert.ToInt16(lblCorrect.Text) + 1 * i);
         lblFalse.Text = Convert.ToString(Convert.ToInt16(lblFalse.Text) - 1 * i);
         foreach (Control c in Controls) {
            if (c.GetType() == tags1.GetType()) {
               if (c.Name.Equals("label" + j.ToString())) {
                  if (!c.Text.Equals("")) {
                     t = Convert.ToInt16(c.Text);
                     switch (t) {
                        case 1:
                           tags1.Text = Convert.ToString(Convert.ToInt16(tags1.Text)
                                                         + 1 * i);
                           ftags1.Text = Convert.ToString(Convert.ToInt16(ftags1.Text)
                                                         - 1 * i);
                           break;
                        case 2:
                           tags2.Text = Convert.ToString(Convert.ToInt16(tags2.Text)
                                                         + 1 * i);
                           ftags2.Text = Convert.ToString(Convert.ToInt16(ftags2.Text)
                                                         - 1 * i);
                           break;
                        case 3:
                           tags3.Text = Convert.ToString(Convert.ToInt16(tags3.Text)
                                                         + 1 * i);
                           ftags3.Text = Convert.ToString(Convert.ToInt16(ftags3.Text)
                                                         - 1 * i);
                           break;
                        case 4:
                           tags4.Text = Convert.ToString(Convert.ToInt16(tags4.Text)
```

```
                                                    + 1 * i);
                        ftags4.Text = Convert.ToString(Convert.ToInt16(ftags4.Text)
                                                    - 1 * i);
                        break;
                     case 5:
                        tags5.Text = Convert.ToString(Convert.ToInt16(tags5.Text)
                                                    + 1 * i);
                        ftags5.Text = Convert.ToString(Convert.ToInt16(ftags5.Text)
                                                    - 1 * i);
                        break;
                     default:
                        tags6.Text = Convert.ToString(Convert.ToInt16(tags6.Text)
                                                    + 1 * i);
                        ftags6.Text = Convert.ToString(Convert.ToInt16(ftags6.Text)
                                                    - 1 * i);
                        break;
                  }
               } else {
                  tags1.Text = Convert.ToString(Convert.ToInt16(tags1.Text)
                                                    + 1 * i);
                  ftags1.Text = Convert.ToString(Convert.ToInt16(ftags1.Text)
                                                    - 1 * i);
               }
            }
         }
      }

      private void checkBox0_CheckedChanged(object sender, EventArgs e) {
         if (checkBox0.Checked) {
            updateFindings(1, 0);
         } else {
            updateFindings(-1, 0);
         }
      }
.
.
.
      private void checkBox49_CheckedChanged(object sender, EventArgs e) {
         if (checkBox49.Checked) {
            updateFindings(1, 49);
         } else {
            updateFindings(-1, 49);
         }
      }

      private void btnReset_Click(object sender, EventArgs e) {
         PictureBox p;
         CheckBox chk;

         foreach (Control c in Controls) {
            if (c.GetType() == checkBox0.GetType()) {
               chk = (CheckBox)c;
               chk.Checked = true;
            } else if (c.GetType() == pictureBox0.GetType()) {
               p = (PictureBox)c;
               p.Image.Dispose();
            }
         }
         lblCorrect.Text = Convert.ToString(50);
         lblFalse.Text = Convert.ToString(0);
         tags1.Text = "0";
         tags2.Text = "0";
         tags3.Text = "0";
         tags4.Text = "0";
         tags5.Text = "0";
         tags6.Text = "0";
         ftags1.Text = "0";
         ftags2.Text = "0";
         ftags3.Text = "0";
         ftags4.Text = "0";
```

```
            ftags5.Text = "0";
            ftags6.Text = "0";
            txtList.Text = "";
        }

        private int countTags(string raw) {
            int count = 1;
            if (raw.Contains(" ")) {
                count += countTags(raw.Substring(raw.IndexOf(" ") + 1));
                return count++;
            }
            return count;
        }

        private Boolean foundAll(string s1, string s2) {
            Boolean f = true;
            foreach (string s in s1.Split(Convert.ToChar(","))) {
                if (!s2.Contains(s.ToString())) {
                    f = false;
                    break;
                }
            }
            return f;
        }

        private void btnSearch1tag_Click(object sender, EventArgs e) {
            PictureBox p;
            Photos Result;
            PhotoSearchOptions searchOptions;
            int counter = 0;
            int piccount = 1;
            int ctags = countTags(txtSearch1.Text);
            string[] txt = txtSearch1.Text.Split(Convert.ToChar(","));

            try {
                searchOptions = new PhotoSearchOptions();
                searchOptions.Tags = txtSearch1.Text;
                searchOptions.PerPage = 50;
                searchOptions.TagMode = TagMode.AllTags;
                Result = flickr.PhotosSearch(searchOptions);
                lblResult.Text = Result.TotalPhotos.ToString();
                foreach (Control c in Controls) {
                    if (c.GetType() == pictureBox0.GetType()) {
                        p = (PictureBox)c;
                        while (!foundAll(txtSearch1.Text,
                            Result.PhotoCollection[counter].RawTags)
                            || (countTags(Result.PhotoCollection[counter].RawTags) > ctags))
{
                            counter++;
                            piccount++;
                            lbltag1.Text = Convert.ToString(piccount);
                            lbltag1.Refresh(); // Refresh();
                            if (piccount == Result.TotalPhotos) {
                                break;
                            }
                            if (counter == 50) {
                                searchOptions.Page++;
                                Result = flickr.PhotosSearch(searchOptions);
                                counter = 0;
                            }
                        }

                        p.Image =
GetImage(Result.PhotoCollection[counter].ThumbnailUrl);
```

```
                txtList.Text +=
Result.PhotoCollection[counter].RawTags + " | ";
                tags1.Text = Convert.ToString(Convert.ToInt16(tags1.Text) + 1);
                Refresh();


                counter++;
                if (counter == 50) {
                   searchOptions.Page++;
                   Result = flickr.PhotosSearch(searchOptions);
                   counter = 0;
                }
                piccount++;
                if (piccount == Result.TotalPhotos) {
                   break;
                }
                lbltag1.Text = Convert.ToString(piccount);
             }
          }
       } catch (Exception ex) { MessageBox.Show(ex.Message); }
    }

    private void btnSearchAll_Click(object sender, EventArgs e) {
       PictureBox p;
       Label l;
       int counter = 0;

       PhotoSearchOptions searchOptions = new PhotoSearchOptions();
       searchOptions.Tags = txtSearch1.Text;
       searchOptions.TagMode = TagMode.AllTags;
       searchOptions.PerPage = 50;
       Photos Result = flickr.PhotosSearch(searchOptions);
       lblResult.Text = Result.TotalPhotos.ToString();
       try {
          foreach (Control c in Controls) {
             if (c.GetType() == pictureBox0.GetType()) {
                p = (PictureBox)c;
                if (p.Name.StartsWith("pictureBox")) {
                   counter = Convert.ToInt16(p.Name.Substring(10));
                   p.Image = GetImage(Result.PhotoCollection[counter].ThumbnailUrl);
                   txtList.Text += Result.PhotoCollection[counter].RawTags + "\n\n";
                   foreach (Control d in Controls) {
                      if (d.GetType() == label0.GetType()) {
                         l = (Label)d;
                         if (l.Name.Equals("label" + counter.ToString())) {
                            l.Text =
countTags(Result.PhotoCollection[counter].RawTags).ToString();
                            switch (l.Text) {
                               case "1":
                                  tags1.Text =
Convert.ToString(Convert.ToInt16(tags1.Text) + 1);
                                  break;
                               case "2":
                                  tags2.Text =
Convert.ToString(Convert.ToInt16(tags2.Text) + 1);
                                  break;
                               case "3":
                                  tags3.Text =
Convert.ToString(Convert.ToInt16(tags3.Text) + 1);
                                  break;
                               case "4":
                                  tags4.Text =
Convert.ToString(Convert.ToInt16(tags4.Text) + 1);
                                  break;
                               case "5":
                                  tags5.Text =
Convert.ToString(Convert.ToInt16(tags5.Text) + 1);
                                  break;
                               default:
```

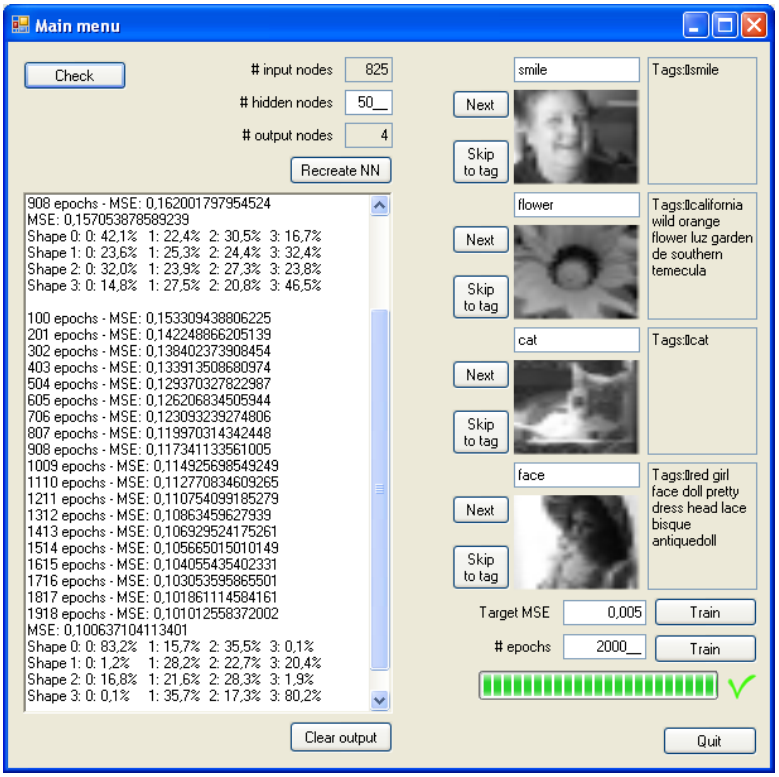
```
                                    tags6.Text =
Convert.ToString(Convert.ToInt16(tags6.Text) + 1);
                                    break;
                                }

                            }
                        }
                    }
                    Refresh();
                }
            }
        }
    } catch (Exception ex) { MessageBox.Show(ex.Message); }
  }

}
```

8.4 Et program som kombinerer MLP og Flickr

I dette programmet har jeg kombinert 8.2 og 8.3. Illustrasjonen viser status etter trening med kun valgte bilder. Som klart vises her er ikke resultatene i seg selv spesielt interessante.

Som nevnt mangler det her både tilgang til folksonomier eller taggeskyer fra Flickr og taksonomi-filtrering ved hjelp av for eksempel WordNet.

Men før det inkluderes må jeg først få det nevrale nettverket til å fungere med mer enn et skjult lag.



Figur 8: GUI til mitt program for henting av fire bilder fra Flickr basert på søk etter tagger. Benyttes deretter til trening av et nevralt nettverk. Figuren viser status etter totalt 3000 epochs.

8.4.1 Den nevrale nettverksklassen

Minimalt videreutviklet fra 8.2.1.

```
public class Weight {
   private double w;
   private double c;
   private double n;
   private Node f;

   public Weight (Node from, Node to, Random r) {
      w = (r.NextDouble() - 0.5) / 3;
      f = from;
   }

   public double get () {
      return f.get() * w;
   }
   public double get_only_w () {
      return w;
   }
```

```
   public void add (double value) {
      w += value;
   }
   public Node from {
      get { return f; }
   }
   public double change {
      get { return c; }
      set { c = value; }
   }
   public double new_w {
      get { return n; }
      set { n = value; }
   }
}

public class Node {
   protected int id;             // for PMML support
   protected double v;           // activation
   protected double net;
   protected double bias;
   protected double delta;
   protected double error;
   protected double bias_delta;
   protected double bed;
   protected Weight[] weights;
   protected double Alfa = 0.1;
   protected double Epsilon = 0.1;
   protected double Bepsilon = 0.1;

   public Node (Random r, int id, Layer prev_layer) {
      Node[] tmp_n;
      this.id = id++;
      this.bias = 0.12;
      if (prev_layer != null) {
         weights = new Weight[prev_layer.nodes.Length];
         for (int i = 0; i < prev_layer.nodes.Length; i++) {
            tmp_n = prev_layer.nodes;
            weights[i] = new Weight(tmp_n[i], this, r);
         }
      }
   }

   public Node () { }         // A "null" reference

   public virtual double get () { return v; }
   public double err {
      get { return error; }
      set { error = value; }
   }

   // backpropagation
   public void compute_weight_change () {
      foreach (Weight w in weights) {
         w.change = delta * w.from.get();
      }
      bed += delta;
```

```
      foreach (Weight w in weights) {
        w.new_w = Alfa * w.new_w + Epsilon * w.change;
        w.add(w.new_w);
        w.change = 0;
      }
      bias_delta = Bepsilon * bed + Alfa * bias_delta;
      bias += bias_delta;
      bed = 0;
    }
}

public class Retinal_Node:Node {
    public Retinal_Node (Random r, int id, Layer prev_layer)
        : base(r, id, prev_layer) { }
    public override double get () {
      return v;
    }
    public void set (double value) {
      this.v = value;
    }
}

public class Hidden_Node:Node {
    public Hidden_Node (Random r, int id, Layer prev_layer)
        : base(r, id, prev_layer) { }
    public override double get () {
      // tanh
      net = bias;
      foreach (Weight w in weights) {
        net += w.get();
      }
      v = Math.Tanh(-net);
      return v;
    }

    // Backpropagation
    public void compute_delta () {
      delta = error * (v * (1 - v));
    }
}

public class Out_Node:Node {
    public Out_Node (Random r, int id, Layer prev_layer)
        : base(r, id, prev_layer) { }
    public override double get () {
      // sigmoid
      net = bias;
      foreach (Weight w in weights) {
        net += w.get();
      }
      v = 1 / (1 + Math.Exp(-net));
      return v;
    }

    // Backpropagation
    public double compute_error (double target) {
      error = target - v;
```

```
      delta = error * (v * (1 - v));

      return error;
    }
    public void compute_sum_delta_error () {
      foreach (Weight w in weights) {
        w.from.err += w.get_only_w() * delta;
      }
    }
}


public class Layer {
  protected Node[] n;

  public Layer (int no_nodes, int id, Layer prev_layer,
                bool last, Random r) {
    n = new Node[no_nodes];
    for (int i = 0; i < no_nodes; i++) {
      if (prev_layer == null) {
        n[i] = new Retinal_Node(r, id++, prev_layer);
      } else if (last) {
        n[i] = new Out_Node(r, id++, prev_layer);
      } else {
        n[i] = new Hidden_Node(r, id++, prev_layer);
      }
    }
  }

  public Node[] nodes {
    get { return n; }
  }
}

public class NN {
  protected Layer[] l;
  Layer prev_layer = null;
  bool last;
  int counter = 0;
  Random r = new Random(unchecked((int)DateTime.Now.Ticks));

  public Layer[] layers {
    get { return l; }
  }

  public NN (int[] structure) {
    l = new Layer[structure.Length];
    for (int i = 0; i < structure.Length; i++) {
      if (i == structure.Length - 1) { last = true; }
      l[i] = new Layer(structure[i], counter, prev_layer,
                       last, r);
      prev_layer = l[i];
      counter += l[i].nodes.Length;
    }
  }

  public double[] activate () {
```

```
      double[] output = new double[l[l.Length - 1].nodes.Length];
      int h;

      // compute hidden layers
      for (h = 1; h < l.Length - 2; h++) {
        for (int i = 0; i < l[h].nodes.Length; i++) {
          l[h].nodes[i].get();
        }
      }

      // compute output layer
      h = l.Length - 1;
      for (int i = 0; i < l[h].nodes.Length; i++) {
        output[i] = l[h].nodes[i].get();
      }

      return output;
    }

    public double update (int[] target) {
      double[] result = activate();
      double SSE = 0;
      double err = 0;
      int i;
      int out_layer = l.Length - 1;
      int last_hidden_layer = out_layer - 1;
      int nodes_out = l[out_layer].nodes.Length;

      // compute output delta
      for (i = 0; i < l[out_layer].nodes.Length; i++) {
        Out_Node n = (Out_Node)l[out_layer].nodes[i];
        err = n.compute_error(target[i]);
        SSE += (err * err) / nodes_out;
      }

      foreach (Out_Node n in l[out_layer].nodes) {
        n.compute_sum_delta_error();
      }
      for (i = last_hidden_layer; i > 0; i--) {
        foreach (Hidden_Node n in l[last_hidden_layer].nodes) {
          n.compute_delta();
        }
      }

      for (i = out_layer; i > 0; i--) {
        foreach (Node n in l[i].nodes) {
          n.compute_weight_change();
        }
      }

      return SSE;
    }
}
```

8.4.2 GUI og styringskode

```
public partial class frmMenu:Form {
```

```
   NN nn;

   //const int height = 100;
   //const int width = 75;
   const int height = 33;
   const int width = 25;
   const int size = height * width;

   int no_input = size;
   double[,] input = new double[4, size];
   int[,] target = { { 1, 0, 0, 0 },
                     { 0, 1, 0, 0 },
                     { 0, 0, 1, 0 },
                     { 0, 0, 0, 1 } };
   int no_target = 4;
   int img_per_page = 25;

   double MSE = 0;
   Flickr flickr;
   Photos[] flickrPictures;
   PhotoSearchOptions[] pictureSearch;
   int current_pic1 = 0;
   int current_pic2 = 0;
   int current_pic3 = 0;
   int current_pic4 = 0;

   public frmMenu () {
      InitializeComponent();
      string apikey = "[redacted]";
      flickr = new Flickr(apikey);
      pictureSearch = new PhotoSearchOptions[4];
      pictureSearch[0] = flickInit(txtPic1.Text);
      pictureSearch[1] = flickInit(txtPic2.Text);
      pictureSearch[2] = flickInit(txtPic3.Text);
      pictureSearch[3] = flickInit(txtPic4.Text);
      flickrPictures = new Photos[4];
      for (int i = 0; i < 4; i++) {
         flickrPictures[i] = flickr.PhotosSearch(pictureSearch[i]);
      }
   }

   private void frmMenu_Load (object sender, EventArgs e) {
      txtInputNodes.Text = no_input.ToString();
      txtOutputNodes.Text = no_target.ToString();
      int[] structure = { Convert.ToInt16(txtInputNodes.Text),
        Convert.ToInt16(txtHidden1.Text),
Convert.ToInt16(txtOutputNodes.Text) };
      nn = new NN(structure);

      pic1.Image =
GetImage(flickrPictures[0].PhotoCollection[current_pic1].ThumbnailUrl
, 0);
      tagsPic1.Text = "Tags:\n" +
flickrPictures[0].PhotoCollection[current_pic1].RawTags;
```

```
     pic2.Image =
GetImage(flickrPictures[1].PhotoCollection[current_pic2].ThumbnailUrl
, 1);
     tagsPic2.Text = "Tags:\n" +
flickrPictures[1].PhotoCollection[current_pic2].RawTags;
     pic3.Image =
GetImage(flickrPictures[2].PhotoCollection[current_pic3].ThumbnailUrl
, 2);
     tagsPic3.Text = "Tags:\n" +
flickrPictures[2].PhotoCollection[current_pic3].RawTags;
     pic4.Image =
GetImage(flickrPictures[3].PhotoCollection[current_pic4].ThumbnailUrl
, 3);
     tagsPic4.Text = "Tags:\n" +
flickrPictures[3].PhotoCollection[current_pic4].RawTags;
   }

   private PhotoSearchOptions flickInit (string search) {
      PhotoSearchOptions tmp = new PhotoSearchOptions();
      tmp.Tags = search;
      tmp.PerPage = img_per_page;
      tmp.TagMode = TagMode.AllTags;
      return tmp;
   }

   public bool ThumbnailCallback () { return false; }

   private Image GetImage (string sURL, int buffer) {
      Image tmp_img;
      Bitmap return_img = new Bitmap(width, height);
      Stream str = null;
      HttpWebRequest wReq = (HttpWebRequest)WebRequest.Create(sURL);
      HttpWebResponse wRes = (HttpWebResponse)(wReq).GetResponse();
      str = wRes.GetResponseStream();

      tmp_img = Image.FromStream(str);

      Image.GetThumbnailImageAbort myCallback = new
Image.GetThumbnailImageAbort(ThumbnailCallback);
      tmp_img = (tmp_img as Bitmap).GetThumbnailImage(width, height,
myCallback, IntPtr.Zero);

      int tmp_c;
      int counter = 0;
      progress.Maximum = tmp_img.Width;
      for (int x = 0; x < tmp_img.Width; x++) {
         for (int y = 0; y < tmp_img.Height; y++) {
            tmp_c = ((tmp_img as Bitmap).GetPixel(x, y).R
                   + (tmp_img as Bitmap).GetPixel(x, y).G
                   + (tmp_img as Bitmap).GetPixel(x, y).B) / 3;
            return_img.SetPixel(x, y, Color.FromArgb(tmp_c, tmp_c,
tmp_c));
            input[buffer, counter] = tmp_c / (double)255;
         }
         progress.Value = x;
      }
```

```
      return return_img;
   }

   private void btnCreateNN_Click (object sender, EventArgs e) {
      int[] structure = { Convert.ToInt16(txtInputNodes.Text),
                          Convert.ToInt16(txtHidden1.Text),
                          Convert.ToInt16(txtOutputNodes.Text) };
      nn = new NN(structure);
   }

   private void btnQuit_Click (object sender, EventArgs e) {
      this.Close();
   }

   private void feed (int i) {
      Retinal_Node tmp_n;
      for (int j = 0; j < 9; j++) {
         tmp_n = (Retinal_Node)nn.layers[0].nodes[j];
         tmp_n.set(input[i, j]);
      }
   }

   private void btnCheck_Click (object sender, EventArgs e) {
      double[] result = new double[8];

      for (int i = 0; i < 4; i++) {
         feed(i);
         txtOut.Text += "Shape " + i.ToString() + ": ";
         result = nn.activate();
         for (int j = 0; j < 4; j++) {
            txtOut.Text += j + ": " + Convert.ToDouble(result[j]
                * 100).ToString("##0.0") + "%\t";
         }
         txtOut.Text += "\n";
      }
      //txtOut.Text += "MSE: " + MSE.ToString("0.0000") + "\n";
      txtOut.Text += "\n";
   }

   private void btnTrain_Click (object sender, EventArgs e) {
      int[] t = new int[4];
      double tmp_mse;
      int counter = 0;

      picOK.Visible = false;
      progress.Maximum = Convert.ToInt16(txtEpochs.Text);
      progress.Value = 0;
      for (int epochs = 0; epochs < Convert.ToInt16(txtEpochs.Text);
epochs++) {
         tmp_mse = 0;
         for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) { t[j] = target[i, j]; }
            feed(i);
            tmp_mse += nn.update(t);
         }
         progress.Value += 1;
         MSE = tmp_mse / 4;
```

```
        if (counter++ == 100) {
          txtOut.Text += epochs + " epochs - MSE: " + MSE + "\n";
          counter = 0;
        }
      }
      txtOut.Text += "MSE: " + MSE + "\n";
      picOK.Visible = true;
    }

    private void btnTargetMSE_Click (object sender, EventArgs e) {
      int[] t = new int[4];
      int counter = 0;
      double tmp_mse;
      MSE = 10;
      picOK.Visible = false;
      progress.Maximum = 1000;
      progress.Value = 1000;
      while (MSE > Convert.ToDouble(txtTargetMSE.Text)) {
        tmp_mse = 0;
        for (int i = 0; i < 4; i++) {
          for (int j = 0; j < 4; j++) { t[j] = target[i, j]; }
          feed(i);
          tmp_mse += nn.update(t);
        }
        MSE = tmp_mse / 4;
        progress.Value = Convert.ToInt16((MSE * 1000));
        counter++;
      }
      picOK.Visible = true;
      txtOut.Text += "Epocs: " + counter.ToString() + "\n";
    }

    private void cmdClear_Click (object sender, EventArgs e) {
      txtOut.Text = "";
    }

    private void btnNext1_Click (object sender, EventArgs e) {
      if (current_pic1 == img_per_page - 1) {
        pictureSearch[0].Page++;
        flickrPictures[0] = flickr.PhotosSearch(pictureSearch[0]);
        current_pic1 = 0;
      }
      pic1.Image =
GetImage(flickrPictures[0].PhotoCollection[++current_pic1].ThumbnailU
rl, 1);
      tagsPic1.Text = "Tags:\n" +
flickrPictures[0].PhotoCollection[current_pic1].RawTags;
    }

    private void btnNext2_Click (object sender, EventArgs e) {
      if (current_pic2 == img_per_page - 1) {
        pictureSearch[1].Page++;
        flickrPictures[1] = flickr.PhotosSearch(pictureSearch[1]);
        current_pic2 = 0;
      }
```

```
      pic2.Image =
GetImage(flickrPictures[1].PhotoCollection[++current_pic2].ThumbnailU
rl, 1);
      tagsPic2.Text = "Tags:\n" +
flickrPictures[1].PhotoCollection[current_pic2].RawTags;
    }

    private void btnNext3_Click (object sender, EventArgs e) {
      if (current_pic3 == img_per_page - 1) {
        pictureSearch[2].Page++;
        flickrPictures[2] = flickr.PhotosSearch(pictureSearch[2]);
        current_pic3 = 0;
      }
      pic3.Image =
GetImage(flickrPictures[2].PhotoCollection[++current_pic3].ThumbnailU
rl, 2);
      tagsPic3.Text = "Tags:\n" +
flickrPictures[2].PhotoCollection[current_pic3].RawTags;
    }

    private void btnNext4_Click (object sender, EventArgs e) {
      if (current_pic4 == img_per_page - 1) {
        pictureSearch[3].Page++;
        flickrPictures[3] = flickr.PhotosSearch(pictureSearch[3]);
        current_pic4 = 0;
      }
      pic4.Image =
GetImage(flickrPictures[3].PhotoCollection[++current_pic4].ThumbnailU
rl, 3);
      tagsPic4.Text = "Tags:\n" +
flickrPictures[3].PhotoCollection[current_pic4].RawTags;
    }

    private void btnSkipTag1_Click (object sender, EventArgs e) {
      btnNext1_Click(null, null);
      try {
        while (txtPic1.Text !=
flickrPictures[0].PhotoCollection[current_pic1].RawTags) {
          btnNext1_Click(null, null);
        }
      } catch (Exception ex) {
        MessageBox.Show(ex.Message);
      }
    }

    private void btnSkipTag2_Click (object sender, EventArgs e) {
      btnNext2_Click(null, null);
      try {
        while (txtPic2.Text !=
flickrPictures[1].PhotoCollection[current_pic2].RawTags) {
          btnNext2_Click(null, null);
        }
      } catch (Exception ex) {
        MessageBox.Show(ex.Message);
      }
    }
```

```
  private void btnSkipTag3_Click (object sender, EventArgs e) {
    btnNext3_Click(null, null);
    try {
      while (txtPic3.Text !=
flickrPictures[2].PhotoCollection[current_pic3].RawTags) {
        btnNext3_Click(null, null);
      }
    } catch (Exception ex) {
      MessageBox.Show(ex.Message);
    }
  }

  private void btnSkipTag4_Click (object sender, EventArgs e) {
    btnNext4_Click(null, null);
    try {
      while (txtPic4.Text !=
flickrPictures[3].PhotoCollection[current_pic4].RawTags) {
        btnNext4_Click(null, null);
      }
    } catch (Exception ex) {
      MessageBox.Show(ex.Message);
    }
  }

}
```